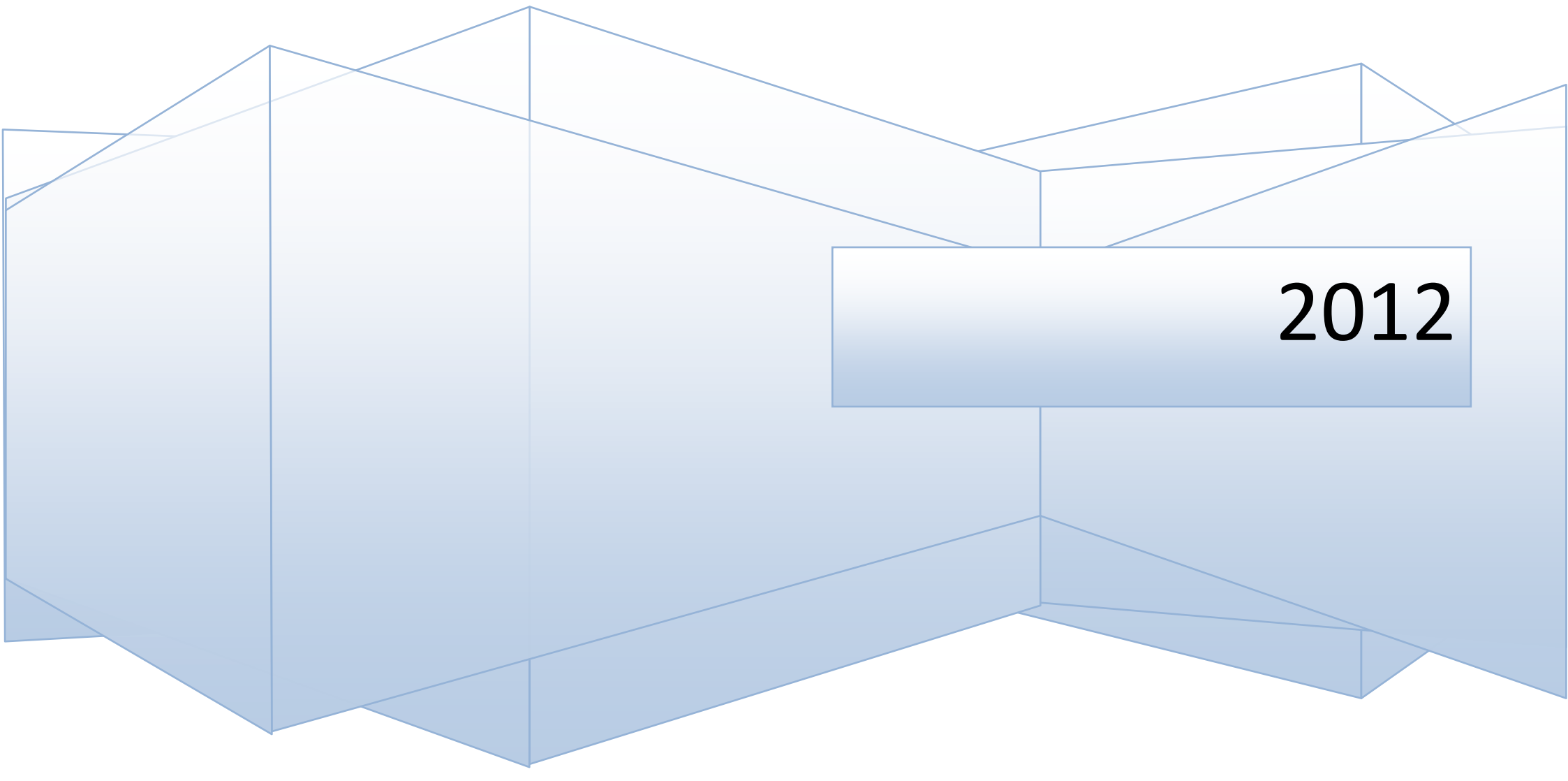
http://darwin.50webs.com

Desarrollo de un evaluador de expresiones algebraicas.

En C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal

Rafael Alberto Moreno Parra



2012

Licencia de este libro

Contenido

Licencia de este libro 2
Introducción 10
Sobre el Autor 10
¿Cómo hacer un evaluador de expresiones algebraicas? 11
Parte 0. Quitar los espacios 12
C++ 12
Visual C# .NET 2010 12
Visual Basic .NET 2010 12
Java 12
PHP 13
JavaScript 13
Object Pascal 13
Parte 1. Chequear que la expresión algebraica no tiene errores de sintaxis 14
1. Cadena Vacía 15
C++ 15
Visual C# .NET 2010 15
Visual Basic .NET 2010 15
Java 15
PHP 15
JavaScript 15
Object Pascal 15
2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3 16
C++ 16
Visual C# .NET 2010 16
Visual Basic .NET 2010 16
Java 17
PHP 17
JavaScript 17
Object Pascal 18
3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 19
C++ 19
Visual C# .NET 2010 19
Visual Basic .NET 2010 19
Java 20
PHP 20
JavaScript 20

Object Pascal 21
4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) 22
C++ 22
Visual C# .NET 2010 22
Visual Basic .NET 2010 22
Java 23
PHP 23
JavaScript 23
Object Pascal 24
5. Que empiece con operador +, *, /, -, ^. Ejemplo: /12-5*2, *17-4 25
C++ 25
Visual C# .NET 2010 25
Visual Basic .NET 2010 25
Java 25
PHP 25
JavaScript 26
Object Pascal 26
6. Que termine con operador. Ejemplo: 12-67*, 2/3- 27
C++ 27
Visual C# .NET 2010 27
Visual Basic .NET 2010 27
Java 27
PHP 28
JavaScript 28
Object Pascal 28
7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) 29
C++ 29
Visual C# .NET 2010 29
Visual Basic .NET 2010 29
Java 30
PHP 30
JavaScript 30
Object Pascal 31
8. Que haya paréntesis vacío. Ejemplo: 2-()*3 32
C++ 32
Visual C# .NET 2010 32
Visual Basic .NET 2010 32
Java 33
PHP 33
JavaScript 33
Object Pascal 34
9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4 35
C++ 35
Visual C# .NET 2010 35
Visual Basic .NET 2010 35
Java 36
PHP 36
JavaScript 36
Object Pascal 37
10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1 38
C++ 38

Visual C# .NET 2010 38
Visual Basic .NET 2010 38
Java 39
PHP 39
JavaScript 39
Object Pascal 40
11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) 41
C++ 41
Visual C# .NET 2010 41
Visual Basic .NET 2010 41
Java 42
PHP 42
JavaScript 42
Object Pascal 43
12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2 44
C++ 44
Visual C# .NET 2010 44
Visual Basic .NET 2010 44
Java 45
PHP 45
JavaScript 45
Object Pascal 46
13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1 47
C++ 47
Visual C# .NET 2010 47
Visual Basic .NET 2010 47
Java 48
PHP 48
JavaScript 48
Object Pascal 49
14. Una variable seguida de un punto. Ejemplo: 4-z.1+3 50
C++ 50
Visual C# .NET 2010 50
Visual Basic .NET 2010 50
Java 51
PHP 51
JavaScript 51
Object Pascal 52
15. Un punto seguido de una variable. Ejemplo: 7-2.p+1 53
C++ 53
Visual C# .NET 2010 53
Visual Basic .NET 2010 53
Java 54
PHP 54
JavaScript 54
Object Pascal 55
16. Un número antes de una variable. Ejemplo: 3x+1 56
C++ 56
Visual C# .NET 2010 56
Visual Basic .NET 2010 56
Java 57

PHP 57
JavaScript 57
Object Pascal 58
17. Un número después de una variable. Ejemplo: x21+4 59
C++ 59
Visual C# .NET 2010 59
Visual Basic .NET 2010 59
Java 60
PHP 60
JavaScript 60
Object Pascal 61
18. Haya caracteres inválidos. Ejemplo: 4+@-1 62
C++ 62
Visual C# .NET 2010 62
Visual Basic .NET 2010 63
Java 63
PHP 63
JavaScript 64
Object Pascal 64
19. Chequea si hay 4 o más variables seguidas 65
C++ 65
Visual C# .NET 2010 65
Visual Basic .NET 2010 65
Java 66
PHP 66
JavaScript 66
Object Pascal 67
20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no 68
C++ 68
Visual C# .NET 2010 69
Visual Basic .NET 2010 70
Java 71
PHP 72
JavaScript 73
Object Pascal 74
21. Si detecta sólo dos letras seguidas es un error. Ejemplo: 4+tk+8.9 75
C++ 75
Visual C# .NET 2010 75
Visual Basic .NET 2010 75
Java 76
PHP 76
JavaScript 76
Object Pascal 77
22. Antes de paréntesis que abre hay una letra. Ejemplo: 4*a(6+b) 78
C++ 78
Visual C# .NET 2010 78
Visual Basic .NET 2010 78
Java 79
PHP 79
JavaScript 79
Object Pascal 80

Llamado de los métodos 81
C++ 81
Visual C# .NET 2010 81
Visual Basic .NET 2010 82
Java 82
PHP 83
JavaScript 83
Object Pascal 84
Interpretación del diagnóstico 85
C++ 85
Visual C# .NET 2010 86
Visual Basic .NET 2010 87
Java 88
PHP 89
JavaScript 90
Object Pascal 91
PARTE 2. EVALUAR UNA EXPRESIÓN SIMPLE (SIN PARÉNTESIS) DE SÓLO NÚMEROS Y LOS OPERADORES BÁSICOS 92
Las piezas 95
C++ 96
Pieza.h 96
Pieza.cpp 96
EvaluadorExpresionAlgebraica.h 97
EvaluadorExpresionAlgebraica.cpp 98
Inicio.cpp 100
Visual C# .NET 2010 101
Pieza.cs 101
EvaluadorExpresionAlgebraica.cs 102
Program.cs 104
Visual Basic .NET 2010 106
Pieza.vb 106
EvaluadorExpresionAlgebraica.vb 107
Module1.vb 109
Java 110
Pieza.java 110
EvaluadorExpresionAlgebraica.java 111
Inicio.java 113
PHP 114
Pieza.php 114
EvaluadorExpresionAlgebraica.php 115
index.php 118
JAVASCRIPT 119
EvaluadorExpresionAlgebraica.js 119
index.html 122
Object Pascal 123
Pieza.pas 123
EvaluadorExpresionAlgebraica.pas 124
PARTE 3. INTERPRETANDO VARIABLES ALGEBRAICAS. 128
Un caso especial: La potencia 129
Las piezas 131
C++ 132
Pieza.h 132

Pieza.cpp 133
EvaluadorExpresionAlgebraica.h 134
EvaluadorExpresionAlgebraica.cpp 135
Principal.cpp 137
Visual C# .NET 139
Pieza.cs 139
EvaluadorExpresionAlgebraica.cs 140
Program.cs 143
Program.cs 144
Visual Basic .NET 145
Pieza.vb 145
EvaluadorExpresionAlgebraica.vb 146
Module.vb 149
Java 150
Pieza.java 150
EvaluadorExpresionAlgebraica.java 151
Inicio.java 154
PHP 155
Pieza.php 155
EvaluadorExpresionAlgebraica.php 156
Index.php 159
JAVASCRIPT 161
EvaluadorExpresionAlgebraica.js 161
index.html 165
Object Pascal 167
Pieza.pas 167
EvaluadorExpresionAlgebraica.pas 169
PARTE 4. INTERPRETACIÓN DE PARÉNTESIS. 174
C++ 175
Pieza.h 175
Pieza.cpp 176
EvaluadorExpresionAlgebraica.h 177
EvaluadorExpresionAlgebraica.cpp 178
C# 182
Pieza.cs 182
EvaluadorExpresionAlgebraica.cs 183
Visual Basic .NET 188
Pieza.vb 188
EvaluadorExpresionAlgebraica.vb 189
Java 193
Pieza.java 193
EvaluadorExpresionAlgebraica.java 194
PHP 199
Pieza.php 199
EvaluadorExpresionAlgebraica.php 200
index.php 204
JAVASCRIPT 205
EvaluadorExpresionAlgebraica.js 205
index.html 210
Object Pascal 211
Pieza.pas 211

EvaluadorExpresionAlgebraica.pas 212
PARTE 5. EVALUADOR COMPLETO. INTERPRETACIÓN DE FUNCIONES ALGEBRAICAS. 217
C++ 217
Pieza.h 217
Pieza.cpp 218
EvaluadorExpresionAlgebraica.h 220
EvaluadorExpresionAlgebraica.cpp 221
Visual C# .NET 226
Pieza.cs 226
EvaluadorExpresionAlgebraica.cs 228
Program.cs 232
Visual Basic .NET 234
Pieza.vb 234
EvaluadorExpresionAlgebraica.vb 235
Module1.vb 239
Java 241
Pieza.java 241
EvaluadorExpresionAlgebraica.java 243
Inicio.java 247
PHP 248
Pieza.php 248
EvaluadorExpresionAlgebraica.php 250
index.php 254
JavaScript 255
EvaluadorExpresionAlgebraica.js 255
index.html 261
Object Pascal 262
Pieza.pas 262
EvaluadorExpresionAlgebraica.pas 264
PARTE 6. PROBANDO EL EVALUADOR DE EXPRESIONES 269
C++ 269
Visual C# 283
Visual Basic .NET 289
Java 293
PHP 297
JavaScript 299
Object Pascal 301
CONCLUSIONES 308

Dedicado a mi familia: José Alberto (mi padre), María del Rosario (mi madre), Diana Pilar (mi hermana) y Sally (mi gata).

Introducción

Tenemos la expresión "5+2*4 - 27/3" almacenada en una variable de tipo Cadena (String). Necesitamos que un algoritmo resuelva esa expresión y retorne el valor cuantitativo que en este caso que es 4.

Las expresiones pueden volverse bastante complejas, por ejemplo:

"3.89-cos(7.12/3.56)+sen(0.02/(1.4-9.5))*1-(5.7/(0.4301+2.891)*.432)"

Y retornar el valor de tipo double 3.56223361 (donde las funciones trigonométricas se evalúan en radianes).

¿Cómo hacer eso? Esa pregunta es respondida paso a paso en este libro.

¿Por qué aprender sobre cómo está hecho un evaluador de expresiones si esa librería o componente se puede conseguir fácilmente en Internet lista para ser usada? Hay varias respuestas a esta pregunta, aquí expongo algunas:

1. Usted es estudiante de Ingeniería de Sistemas o alguna carrera afín en la cual se estudia sobre algoritmos y programación; uno de los temas es el tratamiento de cadenas (Strings) y el evaluador de expresiones es un caso típico de estudio y necesita entender bien el tema.

2. Usted es un desarrollador de software enfocado a los sistemas de información y ha recibido requisitos como por ejemplo: "el software debe calcular la factura telefónica dependiendo si el consumo fue en día normal o en un día festivo o en un día especial (como el día del padre o el día de la madre), en algún horario especial", usualmente este tipo de requisitos para cumplirlos se debe:

    a. Generar el algoritmo.
    b. Implementarlo en el lenguaje de programación de la aplicación original.
    c. Compilar y esperar que la nueva adición no afecte negativamente el código original.
    d. Probar y generar una nueva versión.

    Si el nuevo requisito llegase a cambiar, se debe repetir los pasos a. b. c. y d. con el continuo desgaste y el riesgo de dañar los datos o código de la aplicación original, además que el usuario final requerirá siempre del desarrollador para implementar una nueva idea de facturación para mejorar las ventas.

    En vez de volver a programar, es mucho mejor tener un interpretador de lo que desea el usuario final con la facturación. Entonces el usuario solo ingresaría la nueva fórmula de facturación:

    Valor Factura = (Consumo() * DescuentoDiaEspecial() – BonoRegaloEmpresa() + RecargoServicioEspecial()) * Impuesto()

    Si usted observa, la factura es calculada mediante una expresión algebraica, ¿pero cómo interpretarla si es una cadena (string)? Entender los evaluadores de expresiones algebraicos es un buen inicio para poder implementar evaluadores más complejos usados al interior de un sistema de información. Esta funcionalidad hará que un sistema de información sea bastante flexible, le dé más longevidad e inclusive el usuario final podrá por si mismo hacer los cambios sin tener a un desarrollador siempre al lado.

Sobre el Autor

Rafael Alberto Moreno Parra. Ingeniero de Sistemas. Maestría en Ingeniería con Énfasis en Ingeniería de Sistemas y Computación.

Sitio web: http://darwin.50webs.com y correo: ramsoftware@gmail.com

El evaluador de expresiones es un componente fundamental para la investigación sobre la regresión simbólica, cuyo objetivo es buscar el patrón de una serie de datos. Ese patrón es una expresión algebraica.

El código fuente está listo para descarga aquí: http://darwin.50webs.com/Espanol/Evaluador.htm

¿Cómo hacer un evaluador de expresiones algebraicas?

Yendo por partes. En primer lugar debemos ver que requiere nuestro evaluador de expresiones; este sería un listado:

1. Evaluar los operadores como la suma (+), resta (-), multiplicación (*), división (/) y el exponente (^)
2. Uso de números reales (por ejemplo, 3.74193)
3. Uso de paréntesis. Idealmente que sea ilimitado el número de paréntesis.
4. Uso de funciones como sin(seno), cos(coseno), tan(tangente)
5. Uso de variables que irían de la ‘a’ a la ‘z’
6. Seguir las estrictas normas algebraicas como la precedencia de los operadores.

Parte 0. Quitar los espacios

Para poder hacer el análisis de la sintaxis de la expresión, primero hay que quitar todos los espacios de la cadena.

C++

```cpp
//Retira los espacios de la expresión
void SintaxisExpresionAlgebraica::SinEspacios(char *expr, char *exprOrig)
{
	int pos, posExpr;
	char car1;

	//Va de carácter en carácter
	for(pos = 0, posExpr = 0; *(exprOrig+pos); pos++)
	{
		car1 = exprOrig[pos];
		if (car1 != ' ') expr[posExpr++] = car1;
	}
}
```

Visual C# .NET 2010

```csharp
//Retira los espacios de la expresión
private string SinEspacios(string expr)
{
    string nueva = "";
    int largo, pos;
    char car1;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr[pos];
        if (car1 != ' ') nueva = nueva + car1.ToString();
    }
    return nueva;
}
```

Visual Basic .NET 2010

```vb
'Retira los espacios de la expresión
Private Function SinEspacios(ByVal expr As String) As String
    Dim nueva As String = ""
    Dim largo, pos As Integer
    Dim car1 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 1 Step 1
        car1 = expr(pos)
        If car1 <> " " Then nueva = nueva + car1
    Next
    Return nueva

End Function
```

Java

```java
//Retira los espacios de la expresión
private String SinEspacios(String expr)
{
    String nueva = "";
    int largo, pos;
    char car1;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 != ' ') nueva = nueva + car1;
    }
    return nueva;
}
```

PHP

```php
//Retira los espacios de la expresión
function SinEspacios($expr)
{
     $largo = strlen($expr);
         $nueva = "";

     //Va de carácter en carácter
     for($pos = 0; $pos <= $largo - 1; $pos++)
     {
         $car1 = $expr{$pos};
         if ($car1 != ' ') $nueva = $nueva . $car1;
     }
     return $nueva;
 }
```

JavaScript

```javascript
//Retira los espacios de la expresión
function SinEspacios(expr)
{
     var nueva = "";
     var largo, pos;
     var car1;

     largo = expr.length;

     //Va de carácter en carácter
     for(pos = 0; pos <= largo - 1; pos++)
     {
         car1 = expr.charAt(pos);
         if (car1 != ' ') nueva = nueva + car1;
     }
     return nueva;
}
```

Object Pascal

```
//1. Retira los espacios de la expresión
function TSintaxisExpresionAlgebraica.SinEspacios(expr: string): string;
var
  nueva: string;
  largo, pos: integer;
  car1: char;
begin
  nueva := '';
  largo := length(expr);

  //Va de carácter en carácter
  for pos := 1 to largo do
  begin
      car1 := expr[pos];
      if car1 <> ' ' then nueva := nueva + car1;
  end;
  SinEspacios := nueva;
end;
```

Parte 1. Chequear que la expresión algebraica no tiene errores de sintaxis

Este paso es primordial, porque necesitamos asegurar que los algoritmos que evaluarán la expresión no se bloqueen cuando no encuentren un paréntesis que debe cerrar y este no existiera, o hacer frente a dos o más operadores seguidos. Cuando el usuario digita una expresión algebraica, puede cometer varios errores de sintaxis, luego se deben validar los posibles errores de ingreso.

El listado de posibles errores de sintaxis es el siguiente:

1. Cadena vacía
2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
6. Que termine con operador. Ejemplo: 12-67* 2/3-
7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
8. Que haya paréntesis vacío. Ejemplo: 2-()*3
9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2
13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
16. Un número antes de una variable. Ejemplo: 3x+1 Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
17. Un número después de una variable. Ejemplo: x9+1
18. Hayan caracteres inválidos. Ejemplo: 4+@-1
19. Chequea si hay 4 o más variables seguidas
20. Función inexistente: Ejemplo: xyz(45)
21. Las variables solo son de una sola letra, no debe haber dos letras seguidas o 4 o más letras seguidas. Ejemplo: 3-xy+2*ab
22. Una variable seguida de un paréntesis que abre. Ejemplo: 7-x(5-6)

Luego es necesaria una clase dedicada a esta labor, la de revisar la sintaxis.

1. Cadena Vacía

Una vez retirados los espacios y tabuladores, si la longitud de la cadena es cero entonces tenemos el primer error: una cadena vacía. Los siguientes códigos hacen parte de un método, se ponen aquí para mostrar cómo se valida la cadena vacía. En el punto "Llamado a los métodos" se muestra donde se usan estos códigos:

C++

```
      if (strlen(expr) == 0) { free(expr); return 1; }
```

Visual C# .NET 2010

```
        if (expr.Length == 0) return 1;
```

Visual Basic .NET 2010

```
        If expr.Length = 0 Then Return 1
```

Java

```
        if (expr.length() == 0) return 1;
```

PHP

```
        if (strlen($expr) == 0) return 1;
```

JavaScript

```
        if (expr.length == 0) return 1;
```

Object Pascal

```
        if length(expr) = 0 then begin EvaluaSintaxis :=  1; exit;  end;
```

2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3

El algoritmo debe revisar un carácter y el siguiente, si ambos son operadores entonces es un error.

C++

```
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
bool SintaxisExpresionAlgebraica::DosOperadoresSeguidos(char *expr)
{
	//Va de carácter en carácter
	for (int pos=0; *(expr+pos+1); pos++)
	{
		char car1 = expr[pos];  //Extrae un carácter
		char car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el carácter y el siguiente son operadores, dado el caso retorna true
		if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
			if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
				return true;
	}

	//No encontró doble operador seguido
	return false;
}
```

Visual C# .NET 2010

```
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
private Boolean DosOperadoresSeguidos(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }

    //No encontró doble operador seguido
    return false;
}
```

Visual Basic .NET 2010

```
'2.     Dos o más operadores estén seguidos.  Ejemplo: 2++4, 5-*3
Private Function DosOperadoresSeguidos(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)      'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el carácter y el siguiente son operadores, dado el caso retorna True
        If car1 = "+" Or car1 = "-" Or car1 = "*" Or car1 = "/" Or car1 = "^" Then
            If car2 = "+" Or car2 = "-" Or car2 = "*" Or car2 = "/" Or car2 = "^" Then Return True
        End If
    Next

    'No encontró doble operador seguido
    Return False
End Function
```

Java

```java
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
private Boolean DosOperadoresSeguidos(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }

    //No encontró doble operador seguido
    return false;
}
```

PHP

```php
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function DosOperadoresSeguidos($expresion)
{
      $largo = strlen($expresion);

      //Va de carácter en carácter
      for ($pos = 0; $pos < $largo - 1; $pos++)
      {
            $car1 = $expresion{$pos}; //Extrae un carácter
            $car2 = $expresion{$pos + 1}; //Extrae el siguiente carácter

            //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
            if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
                  if ($car2 == '+' || $car2 == '-' || $car2 == '*' || $car2 == '/' || $car2 == '^')
                        return true;
      }

      //No encontró doble operador seguido
      return false;
}
```

JavaScript

```javascript
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function DosOperadoresSeguidos(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }

    //No encontró doble operador seguido
    return false;
}
```

Object Pascal

```
//2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function TSintaxisExpresionAlgebraica.DosOperadoresSeguidos(expr: string): boolean;
var
  largo, pos: integer;
  car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
            if (car2 = '+') or (car2 = '-') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
            begin
               DosOperadoresSeguidos := true;
               exit;
            end;
    end;

    //No encontró doble operador seguido
    DosOperadoresSeguidos := false;
end;
```

3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7

El algoritmo chequea si un carácter es operador y el siguiente es un paréntesis que cierra.

C++

```
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
bool SintaxisExpresionAlgebraica::OperadorParentesisCierra(char *expr)
{
      int pos;
      char car1, car2;

      //Va de carácter en carácter
      for(pos = 0; *(expr+pos+1); pos++)
      {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos+1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
                  if (car2 == ')') return true;
      }

      //No encontró operador seguido de un paréntesis que cierra
      return false;
}
```

Visual C# .NET 2010

```
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
private Boolean OperadorParentesisCierra(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == ')') return true;
    }

    //No encontró operador seguido de un paréntesis que cierra
    return false;
}
```

Visual Basic .NET 2010

```
'3. Un operador seguido de un paréntesis que cierra.  Ejemplo: 2-(4+)-7
Private Function OperadorParentesisCierra(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)      'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        If car1 = "+" Or car1 = "-" Or car1 = "*" Or car1 = "/" Or car1 = "^" Then
            If car2 = ")" Then Return True
        End If
    Next

    'No encontró operador seguido de un paréntesis que cierra
    Return False
End Function
```

Java

```java
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
private Boolean OperadorParentesisCierra(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == ')') return true;
    }

    //No encontró operador seguido de un paréntesis que cierra
    return false;
}
```

PHP

```php
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function OperadorParentesisCierra($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
            if ($car2 == ')') return true;
    }

    //No encontró operador seguido de un paréntesis que cierra
    return false;
}
```

JavaScript

```javascript
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function OperadorParentesisCierra(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == ')') return true;
    }

    //No encontró operador seguido de un paréntesis que cierra
    return false;
}
```

Object Pascal

```
//3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function TSintaxisExpresionAlgebraica.OperadorParentesisCierra(expr: string): boolean;
var
  largo, pos: integer;
  car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
            if (car2 = ')') then
            begin
              OperadorParentesisCierra := true;
              exit;
            end;
    end;

    //No encontró operador seguido de un paréntesis que cierra
    OperadorParentesisCierra := false;
end;
```

4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)

El algoritmo chequea si un carácter es un paréntesis que abre y el siguiente es un operador. Después de un paréntesis que abre podría seguir el "-", ese menos unario no será tenido en cuenta en el evaluador de expresiones que se construirá, luego en aquellas expresiones que después de paréntesis que abre y sigue el "-", se debe anteponer el número cero. Ejemplo: 4*(-x+3) se debe escribir 4*(0-x+3)

C++

```
//4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
bool SintaxisExpresionAlgebraica::ParentesisAbreOperador(char *expr)
{
      int pos;
      char car1, car2;

      //Va de carácter en carácter
      for(pos = 0; *(expr+pos+1); pos++)
      {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos+1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if (car1 == '(')
                  if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
      }

      //No encontró paréntesis que abre seguido de un operador
      return false;
}
```

Visual C# .NET 2010

```
    //4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    private Boolean ParentesisAbreOperador(string expr)
    {
        int largo, pos;
        char car1, car2;

        largo = expr.Length;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 2; pos++)
        {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if (car1 == '(')
                if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
        }

        //No encontró paréntesis que abre seguido de un operador
        return false;
    }
```

Visual Basic .NET 2010

```
    '4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    Private Function ParentesisAbreOperador(ByVal expr As String) As Boolean
        Dim largo, pos As Integer
        Dim car1, car2 As Char

        largo = expr.Length

        'Va de carácter en carácter
        For pos = 0 To largo - 2 Step 1
            car1 = expr(pos)     'Extrae un carácter
            car2 = expr(pos + 1) 'Extrae el siguiente carácter

            'Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            If car1 = "(" Then
                If car2 = "+" Or car2 = "-" Or car2 = "*" Or car2 = "/" Or car2 = "^" Then Return True
            End If
        Next

        'No encontró paréntesis que abre seguido de un operador
        Return False
    End Function
```

Java

```java
//4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
private Boolean ParentesisAbreOperador(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if (car1 == '(')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }

    //No encontró paréntesis que abre seguido de un operador
    return false;
}
```

PHP

```php
//4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function ParentesisAbreOperador($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if ($car1 == '(')
            if ($car2 == '+' || $car2 == '-' || $car2 == '*' || $car2 == '/' || $car2 == '^') return true;
    }

    //No encontró paréntesis que abre seguido de un operador
    return false;
}
```

JavaScript

```javascript
//4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function ParentesisAbreOperador(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if (car1 == '(')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }

    //No encontró paréntesis que abre seguido de un operador
    return false;
}
```

Object Pascal

```
//4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function TSintaxisExpresionAlgebraica.ParentesisAbreOperador(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1  do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if (car1 = '(') then
            if (car2 = '+') or (car2 = '-') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
            begin
                 ParentesisAbreOperador := true;
                 exit;
            end;
    end;

    //No encontró paréntesis que abre seguido de un operador
    ParentesisAbreOperador := false;
end;
```

5. Que empiece con operador +, *, /, -, ^. Ejemplo: /12-5*2, *17-4

El algoritmo revisa si el primer carácter es un operador. Una expresión algebraica podría empezar con "-", ese menos unario no será tenido en cuenta en el evaluador de expresiones que se construirá, luego en aquellas expresiones que empiezan con "-", se debe anteponer el número cero. Ejemplo: -x+3 se debe escribir 0-x+3

C++

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
bool SintaxisExpresionAlgebraica::IniciaOperador(char *expr)
{
      char car1;

      //Extrae el primer carácter
      car1 = expr[0];

      //Compara si el carácter es operador
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

      //No encontró que empiece con operador
      return false;
}
```

Visual C# .NET 2010

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
private Boolean IniciaOperador(string expr)
{
    char car1;

    //Extrae el primer carácter
    car1 = expr[0];

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que empiece con operador
    return false;
}
```

Visual Basic .NET 2010

```
'5.     Que empiece con operador +, *, /, -, ^. Ejemplo: /12-5*2 ,  *17-4
Private Function IniciaOperador(ByVal expr As String) As Boolean
    Dim car1 As Char

    'Extrae el primer carácter
    car1 = expr(0)

    'Compara si el carácter es operador
    If car1 = "+" Or car1 = "-" Or car1 = "*" Or car1 = "/" Or car1 = "^" Then Return True

    'No encontró que empiece con operador
    Return False
End Function
```

Java

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
private Boolean IniciaOperador(String expr)
{
    char car1;

    //Extrae el primer carácter
    car1 = expr.charAt(0);

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que empiece con operador
    return false;
}
```

PHP

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
function IniciaOperador($expr)
{
    //Extrae el primer carácter
    $car1 = $expr{0};

    //Compara si el carácter es operador
    if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^') return true;

    //No encontró que empiece con operador
    return false;
}
```

JavaScript

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
function IniciaOperador(expr)
{
    var car1;

    //Extrae el primer carácter
    car1 = expr.charAt(0);

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que empiece con operador
    return false;
}
```

Object Pascal

```
//5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4
function TSintaxisExpresionAlgebraica.IniciaOperador(expr: string): boolean;
var
  car1: char;
begin
    //Extrae el primer carácter
    car1 := expr[1];

    //Compara si el carácter es operador
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      IniciaOperador := true
    else
      //No encontró que empiece con operador
      IniciaOperador := false;
end;
```

6. Que termine con operador. Ejemplo: 12-67*, 2/3-

El algoritmo revisa si el último carácter es un operador

C++

```
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
bool SintaxisExpresionAlgebraica::FinalizaOperador(char *expr)
{
	char car1;
	int largo;

	largo = strlen(expr);
	car1 = expr[largo - 1];

	//Compara si el carácter es operador
	if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

	//No encontró que termine con operador
	return false;
}
```

Visual C# .NET 2010

```
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
private Boolean FinalizaOperador(string expr)
{
    char car1;
    int largo;

    largo = expr.Length;
    car1 = expr[largo - 1];

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que termine con operador
    return false;
}
```

Visual Basic .NET 2010

```
'6.     Que termine con operador. Ejemplo: 12-67*  2/3-
Private Function FinalizaOperador(ByVal expr As String) As Boolean
    Dim car1 As Char
    Dim largo As Integer

    largo = expr.Length
    car1 = expr(largo - 1)

    'Compara si el carácter es operador
    If car1 = "+" Or car1 = "-" Or car1 = "*" Or car1 = "/" Or car1 = "^" Then Return True

    'No encontró que termine con operador
    Return False
End Function
```

Java

```
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
private Boolean FinalizaOperador(String expr)
{
    char car1;
    int largo;

    largo = expr.length();
    car1 = expr.charAt(largo - 1);

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que termine con operador
    return false;
}
```

PHP

```php
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
function FinalizaOperador($expr)
{
    $largo = strlen($expr);
    $car1 = $expr{$largo - 1};

    //Compara si el carácter es operador
    if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^') return true;

    //No encontró que termine con operador
    return false;
}
```

JavaScript

```javascript
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
function FinalizaOperador(expr)
{
    var car1;
    var largo;

    largo = expr.length;
    car1 = expr.charAt(largo - 1);

    //Compara si el carácter es operador
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^') return true;

    //No encontró que termine con operador
    return false;
}
```

Object Pascal

```
//6.Que termine con operador. Ejemplo: 12-67* 2/3-
function TSintaxisExpresionAlgebraica.FinalizaOperador(expr: string): boolean;
var
  car1: char;
  largo: integer;
begin
    largo := length(expr);
    car1 := expr[largo];

    //Compara si el carácter es operador
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
       FinalizaOperador := true
    else
      //No encontró que termine con operador
      FinalizaOperador := false;
end;
```

7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))

El algoritmo cuenta el número de paréntesis que abren "(" con el número de paréntesis que cierran ")", debe ser el mismo número.

C++

```
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
bool SintaxisExpresionAlgebraica::ParentesisDesbalanceados(char *expr)
{
	char car1;
	int parabre, parcierra;

	//Va de carácter en carácter
	parabre = 0;
	parcierra = 0;

	for(int pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];
		if (car1 == '(') parabre++;
		if (car1 == ')') parcierra++;
	}

	if (parabre != parcierra) return true;

	return false;
}
```

Visual C# .NET 2010

```
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
private Boolean ParentesisDesbalanceados(string expr)
{
    char car1;
    int largo, parabre, parcierra;

    //Va de carácter en carácter
    largo = expr.Length;
    parabre = 0;
    parcierra = 0;

    for(int pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr[pos];
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }

    if (parabre != parcierra) return true;

    return false;
}
```

Visual Basic .NET 2010

```
'7.     Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
Private Function ParentesisDesbalanceados(ByVal expr As String) As Boolean
    Dim car1 As Char
    Dim largo, parabre, parcierra As Integer

    'Va de carácter en carácter
    largo = expr.Length
    parabre = 0
    parcierra = 0
    For pos = 0 To largo - 1 Step 1
        car1 = expr(pos)
        If car1 = "(" Then parabre = parabre + 1
        If car1 = ")" Then parcierra = parcierra + 1
    Next

    If parabre <> parcierra Then Return True

    Return False
End Function
```

Java

```java
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
private Boolean ParentesisDesbalanceados(String expr)
{
    char car1;
    int largo, parabre, parcierra;

    //Va de carácter en carácter
    largo = expr.length();
    parabre = 0;
    parcierra = 0;

    for(int pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }

    if (parabre != parcierra) return true;

    return false;
}
```

PHP

```php
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function ParentesisDesbalanceados($expr)
{
    //Va de carácter en carácter
    $largo = strlen($expr);
    $parabre = 0;
    $parcierra = 0;

    for($pos = 0; $pos <= $largo - 1; $pos++)
    {
        $car1 = $expr{$pos};
        if ($car1 == '(') $parabre++;
        if ($car1 == ')') $parcierra++;
    }

    if ($parabre != $parcierra) return true;

    return false;
}
```

JavaScript

```javascript
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function ParentesisDesbalanceados(expr)
{
    var car1;
    var largo, parabre, parcierra;

    //Va de carácter en carácter
    largo = expr.length;
    parabre = 0;
    parcierra = 0;

    for(var pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }

    if (parabre != parcierra) return true;

    return false;
}
```

Object Pascal

```
//7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function TSintaxisExpresionAlgebraica.ParentesisDesbalanceados(expr: string): boolean;
var
  car1: char;
  largo, parabre, parcierra, pos: integer;
begin
    //Va de carácter en carácter
    largo := length(expr);
    parabre := 0;
    parcierra := 0;

    for pos := 1 to largo do
    begin
        car1 := expr[pos];
        if (car1 = '(') then parabre := parabre+1;
        if (car1 = ')') then parcierra := parcierra+1;
    end;

    if (parabre <> parcierra) then
      ParentesisDesbalanceados := true
    else
    ParentesisDesbalanceados := false;
end;
```

8. Que haya paréntesis vacío. Ejemplo: 2-()*3

El algoritmo chequea si el carácter es un paréntesis que abre y el siguiente es un paréntesis que cierra entonces hay un error.

C++

```
//8.Que haya paréntesis vacío. Ejemplo: 2-()*3
bool SintaxisExpresionAlgebraica::ParentesisVacio(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];	//Extrae un carácter
		car2 = expr[pos+1];	//Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
		if (car1 == '(' && car2 == ')') return true;
	}

	//No encontró paréntesis que abre seguido de un paréntesis que cierra
	return false;
}
```

Visual C# .NET 2010

```
//8.	Que haya paréntesis vacío. Ejemplo: 2-()*3
private Boolean ParentesisVacio(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];       //Extrae un carácter
        car2 = expr[pos + 1];   //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == '(' && car2 == ')') return true;
    }

    //No encontró paréntesis que abre seguido de un paréntesis que cierra
    return false;
}
```

Visual Basic .NET 2010

```
'8.	Que haya paréntesis vacío. Ejemplo:  2-()*3
Private Function ParentesisVacio(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)      'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        If car1 = "(" And car2 = ")" Then Return True
    Next

    'No encontró paréntesis que abre seguido de un paréntesis que cierra
    Return False

End Function
```

Java

```java
//8.    Que haya paréntesis vacío. Ejemplo: 2-()*3
private Boolean ParentesisVacio(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);        //Extrae un carácter
        car2 = expr.charAt(pos + 1);    //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == '(' && car2 == ')') return true;
    }

    //No encontró paréntesis que abre seguido de un paréntesis que cierra
    return false;
}
```

PHP

```php
//8. Que haya paréntesis vacío. Ejemplo: 2-()*3
function ParentesisVacio($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};        //Extrae un carácter
        $car2 = $expr{$pos + 1};    //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if ($car1 == '(' && $car2 == ')') return true;
    }

    //No encontró paréntesis que abre seguido de un paréntesis que cierra
    return false;
}
```

JavaScript

```javascript
//8.    Que haya paréntesis vacío. Ejemplo: 2-()*3
function ParentesisVacio(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);        //Extrae un carácter
        car2 = expr.charAt(pos + 1);    //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == '(' && car2 == ')') return true;
    }

    //No encontró paréntesis que abre seguido de un paréntesis que cierra
    return false;
}
```

Object Pascal

```
    //8.    Que haya paréntesis vacío. Ejemplo: 2-()*3
    function TSintaxisExpresionAlgebraica.ParentesisVacio(expr: string): boolean;
    var
        largo, pos: integer;
        car1, car2: char;
    begin
        largo := length(expr);

        //Va de carácter en carácter
        for pos := 1 to largo - 1 do
        begin
            car1 := expr[pos];       //Extrae un carácter
            car2 := expr[pos + 1];   //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
            if (car1 = '(') and (car2 = ')') then
            begin
              ParentesisVacio := true;
              exit;
            end;
        end;
    end;
```

9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4

El algoritmo chequea que exista la correspondencia entre paréntesis que abre con paréntesis que cierra.

C++

```
//9.Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
bool SintaxisExpresionAlgebraica::ParentesisBalanceIncorrecto(char *expr)
{
	int pos, balance;
	char car1;


	balance = 0;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];	//Extrae un carácter
		if (car1 == '(') balance++;
		if (car1 == ')') balance--;
		if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
	}

	return false;
}
```

Visual C# .NET 2010

```
    //9.    Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
    private Boolean ParentesisBalanceIncorrecto(string expr)
    {
        int largo, pos, balance;
        char car1;

        largo = expr.Length;
        balance = 0;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 1; pos++)
        {
            car1 = expr[pos];   //Extrae un carácter
            if (car1 == '(') balance++;
            if (car1 == ')') balance--;
            if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
        }

        return false;
    }
```

Visual Basic .NET 2010

```
    '9.     Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
    Private Function ParentesisBalanceIncorrecto(ByVal expr As String) As Boolean
        Dim largo, pos, balance As Integer
        Dim car1 As Char

        largo = expr.Length
        balance = 0

        'Va de carácter en carácter
        For pos = 0 To largo - 1 Step 1
            car1 = expr(pos)      'Extrae un carácter
            If car1 = "(" Then balance = balance + 1
            If car1 = ")" Then balance = balance - 1
            If balance < 0 Then Return True 'Si cae por debajo de cero es que el balance es erróneo
        Next

        Return False
    End Function
```

Java

```java
//9.    Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
private Boolean ParentesisBalanceIncorrecto(String expr)
{
    int largo, pos, balance;
    char car1;

    largo = expr.length();
    balance = 0;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }

    return false;
}
```

PHP

```php
  //9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
function ParentesisBalanceIncorrecto($expr)
{
    $largo = strlen($expr);
    $balance = 0;

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 1; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        if ($car1 == '(') $balance++;
        if ($car1 == ')') $balance--;
        if ($balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }

    return false;
}
```

JavaScript

```javascript
//9.    Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
function ParentesisBalanceIncorrecto(expr)
{
    var largo, pos, balance;
    var car1;

    largo = expr.length;
    balance = 0;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }

    return false;
}
```

Object Pascal

```
    //9.    Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
    function TSintaxisExpresionAlgebraica.ParentesisBalanceIncorrecto(expr: string): boolean;
    var
        largo, pos, balance: integer;
        car1: char;
    begin
        largo := length(expr);
        balance := 0;

        //Va de carácter en carácter
        for pos := 1 to largo do
        begin
            car1 := expr[pos];   //Extrae un carácter
            if car1 = '(' then balance := balance + 1;
            if car1 = ')' then balance := balance - 1;
            if balance < 0 then
            begin
              ParentesisBalanceIncorrecto := true; //Si cae por debajo de cero es que el balance es erróneo
              exit;
            end;
        end;

        ParentesisBalanceIncorrecto := false;
    end;
```

10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1

El algoritmo chequea que si el carácter es un paréntesis que cierra y luego sigue un número, es un error.

C++

```
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
bool SintaxisExpresionAlgebraica::ParentesisCierraNumero(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
		if (car1 == ')')
			if ( (car2 >= '0' && car2 <= '9') || car2 == '.') return true;
	}

	//No encontró un paréntesis que cierra seguido de un número
	return false;
}
```

Visual C# .NET 2010

```
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
private Boolean ParentesisCierraNumero(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == ')')
            if ( (car2 >= '0' && car2 <= '9') || car2 == '.') return true;
    }

    //No encontró un paréntesis que cierra seguido de un número
    return false;
}
```

Visual Basic .NET 2010

```
'10.    Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
Private Function ParentesisCierraNumero(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)     'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        If car1 = ")" Then
            If (car2 >= "0" And car2 <= "9") Or car2 = "." Then Return True
        End If
    Next

    'No encontró un paréntesis que cierra seguido de un número
    Return False
End Function
```

Java

```java
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
private Boolean ParentesisCierraNumero(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == ')')
            if ( (car2 >= '0' && car2 <= '9') || car2 == '.') return true;
    }

    //No encontró un paréntesis que cierra seguido de un número
    return false;
}
```

PHP

```php
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
function ParentesisCierraNumero($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if ($car1 == ')')
            if ( ($car2 >= '0' && $car2 <= '9') || $car2 == '.') return true;
    }

    //No encontró un paréntesis que cierra seguido de un número
    return false;
}
```

JavaScript

```javascript
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
function ParentesisCierraNumero(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (car1 == ')')
            if ( (car2 >= '0' && car2 <= '9') || car2 == '.') return true;
    }

    //No encontró un paréntesis que cierra seguido de un número
    return false;
```

Object Pascal

```
//10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1
function TSintaxisExpresionAlgebraica.ParentesisCierraNumero(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if car1 = ')' then
            if (car2 = '.') or (car2 in ['0'..'9']) then
            begin
              ParentesisCierraNumero := true;
              exit;
            end;
    end;

    //No encontró paréntesis que abre seguido de un paréntesis que cierra
    ParentesisCierraNumero := false;
end;
```

11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)

El algoritmo chequea que si el carácter es un número y luego sigue un paréntesis que abre, es un error.

C++

```
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
bool SintaxisExpresionAlgebraica::NumeroParentesisAbre(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es número y el siguiente es paréntesis que abre
		if ((car1 >= '0' && car1 <= '9') || car1 == '.')
			if (car2 == '(') return true;
	}

	//No encontró un número seguido de un paréntesis que abre
	return false;
}
```

Visual C# .NET 2010

```
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
private Boolean NumeroParentesisAbre(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if ((car1 >= '0' && car1 <= '9') || car1 == '.')
            if (car2 == '(') return true;
    }

    //No encontró un número seguido de un paréntesis que abre
    return false;
}
```

Visual Basic .NET 2010

```
'11.    Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
Private Function NumeroParentesisAbre(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)     'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es número y el siguiente es paréntesis que abre
        If (car1 >= "0" And car1 <= "9") Or car1 = "." Then
            If car2 = "(" Then Return True
        End If
    Next

    'No encontró un número seguido de un paréntesis que abre
    Return False
End Function
```

Java

```java
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
private Boolean NumeroParentesisAbre(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if ((car1 >= '0' && car1 <= '9') || car1 == '.')
            if (car2 == '(') return true;
    }

    //No encontró un número seguido de un paréntesis que abre
    return false;
}
```

PHP

```php
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function NumeroParentesisAbre($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if ( ($car1 >= '0' && $car1 <= '9') || $car1 == '.' )
            if ($car2 == '(') return true;
    }

    //No encontró un número seguido de un paréntesis que abre
    return false;
}
```

JavaScript

```javascript
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function NumeroParentesisAbre(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if ( (car1 >= '0' && car1 <= '9') || car1 == '.' )
            if (car2 == '(') return true;
    }

    //No encontró un número seguido de un paréntesis que abre
    return false;
}
```

Object Pascal

```
//11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function TSintaxisExpresionAlgebraica.NumeroParentesisAbre(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if (car1 = '.') or (car1 in ['0'..'9']) then
            if car2 = '(' then
            begin
              NumeroParentesisAbre := true;
              exit;
            end;
    end;

    //No encontró un número seguido de un paréntesis que abre
    NumeroParentesisAbre := false;
end;
```

12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2

El algoritmo chequea que un número real no tenga dos puntos.

C++

```
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
bool SintaxisExpresionAlgebraica::DoblePuntoNumero(char *expr)
{
	int pos, totalpuntos;
	char car1;

	totalpuntos = 0;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];	//Extrae un carácter
		if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
		if (car1 == '.') totalpuntos++;
		if (totalpuntos > 1) return true;
	}

	//No encontró Doble punto en un número de tipo real
	return false;
}
```

Visual C# .NET 2010

```
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
private Boolean DoblePuntoNumero(string expr)
{
    int largo, pos, totalpuntos;
    char car1;

    largo = expr.Length;
    totalpuntos = 0;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }

    //No encontró Doble punto en un número de tipo real
    return false;
}
```

Visual Basic .NET 2010

```
'12.    Doble punto en un número de tipo real. Ejemplo: 3-2..4+1   7-6.46.1+2
Private Function DoblePuntoNumero(ByVal expr As String) As Boolean
    Dim largo, pos, totalpuntos As Integer
    Dim car1 As Char

    largo = expr.Length
    totalpuntos = 0

    'Va de carácter en carácter
    For pos = 0 To largo - 1 Step 1
        car1 = expr(pos)     'Extrae un carácter
        If (car1 < "0" Or car1 > "9") And car1 <> "." Then totalpuntos = 0
        If car1 = "." Then totalpuntos = totalpuntos + 1
        If totalpuntos > 1 Then Return True
    Next

    'No encontró Doble punto en un número de tipo real
    Return False
End Function
```

Java

```java
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
private Boolean DoblePuntoNumero(String expr)
{
    int largo, pos, totalpuntos;
    char car1;

    largo = expr.length();
    totalpuntos = 0;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }

    //No encontró Doble punto en un número de tipo real
    return false;
}
```

PHP

```php
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function DoblePuntoNumero($expr)
{
    $largo = strlen($expr);
    $totalpuntos = 0;

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 1; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        if (($car1 < '0' || $car1 > '9') && $car1 != '.') $totalpuntos = 0;
        if ($car1 == '.') $totalpuntos++;
        if ($totalpuntos > 1) return true;
    }

    //No encontró Doble punto en un número de tipo real
    return false;
}
```

JavaScript

```javascript
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function DoblePuntoNumero(expr)
{
    var largo, pos, totalpuntos;
    var car1;

    largo = expr.length;
    totalpuntos = 0;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }

    //No encontró Doble punto en un número de tipo real
    return false;
}
```

Object Pascal

```
//12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function TSintaxisExpresionAlgebraica.DoblePuntoNumero(expr: string): boolean;
var
    largo, pos, totalpuntos: integer;
    car1: char;
begin
    largo := length(expr);
    totalpuntos := 0;

    //Va de carácter en carácter
    for pos := 1 to largo do
    begin
        car1 := expr[pos];   //Extrae un carácter
        if (car1 <> '.') and not(car1 in ['0'..'9']) then totalpuntos := 0;
        if (car1 = '.') then totalpuntos := totalpuntos + 1;
        if (totalpuntos > 1) then
        begin
          DoblePuntoNumero := true;
          exit;
        end;
    end;

    //No encontró Doble punto en un número de tipo real
    DoblePuntoNumero := false;
end;
```

13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1

El algoritmo chequea que si el carácter es un paréntesis que cierra y lo que sigue es una letra entonces es un error.

C++

```
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
bool SintaxisExpresionAlgebraica::ParentesisCierraVariable(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
		if (car1 == ')')
		if (car2 >= 'a' && car2 <= 'z') return true;
	}

	//No encontró un paréntesis que cierra seguido de una variable
	return false;
}
```

Visual C# .NET 2010

```
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
private Boolean ParentesisCierraVariable(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == ')')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un paréntesis que cierra seguido de una variable
    return false;
}
```

Visual Basic .NET 2010

```
'13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
Private Function ParentesisCierraVariable(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)     'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        If car1 = ")" Then
            If car2 >= "a" And car2 <= "z" Then Return True
        End If
    Next

    'No encontró un paréntesis que cierra seguido de una variable
    Return False
End Function
```

Java

```java
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
private Boolean ParentesisCierraVariable(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == ')')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un paréntesis que cierra seguido de una variable
    return false;
}
```

PHP

```php
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function ParentesisCierraVariable($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if ($car1 == ')')
            if ($car2 >= 'a' && $car2 <= 'z') return true;
    }

    //No encontró un paréntesis que cierra seguido de una variable
    return false;
}
```

JavaScript

```javascript
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function ParentesisCierraVariable(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == ')')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un paréntesis que cierra seguido de una variable
    return false;
}
```

Object Pascal

```
//13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function TSintaxisExpresionAlgebraica.ParentesisCierraVariable(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 = ')') then
            if (car2 in ['a'..'z']) then
            begin
              ParentesisCierraVariable := true;
              exit;
            end;
    end;

    //No encontró un número seguido de un paréntesis que abre
    ParentesisCierraVariable := false;
end;
```

14. Una variable seguida de un punto. Ejemplo: 4-z.1+3

El algoritmo chequea que si el carácter es una letra y lo que sigue es un punto entonces es un error.

C++

```
//14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
bool SintaxisExpresionAlgebraica::VariableluegoPunto(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
		if (car1 >= 'a' && car1 <= 'z')
			if (car2 == '.') return true;
	}

	//No encontró una variable seguida de un número
	return false;
}
```

Visual C# .NET 2010

```
    //14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
    private Boolean VariableluegoPunto(string expr)
    {
        int largo, pos;
        char car1, car2;

        largo = expr.Length;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 2; pos++)
        {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 >= 'a' && car1 <= 'z')
                if (car2 == '.') return true;
        }

        //No encontró una variable seguida de un número
        return false;
    }
```

Visual Basic .NET 2010

```
    '14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
    Private Function VariableluegoPunto(ByVal expr As String) As Boolean
        Dim largo, pos As Integer
        Dim car1, car2 As Char

        largo = expr.Length

        'Va de carácter en carácter
        For pos = 0 To largo - 2 Step 1
            car1 = expr(pos)     'Extrae un carácter
            car2 = expr(pos + 1) 'Extrae el siguiente carácter

            'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            If car1 >= "a" And car1 <= "z" Then
                If car2 = "." Then Return True
            End If
        Next

        'No encontró una variable seguida de un número
        Return False
    End Function
```

Java

```java
//14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
private Boolean VariableluegoPunto(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 >= 'a' && car1 <= 'z')
            if (car2 == '.') return true;
    }

    //No encontró una variable seguida de un número
    return false;
}
```

PHP

```php
//14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
function VariableluegoPunto($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if ($car1 >= 'a' && $car1 <= 'z')
            if ($car2 == '.') return true;
    }

    //No encontró una variable seguida de un número
    return false;
}
```

JavaScript

```javascript
//14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
function VariableluegoPunto(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 >= 'a' && car1 <= 'z')
            if (car2 == '.') return true;
    }

    //No encontró una variable seguida de un número
    return false;
}
```

Object Pascal

```
//14. Una variable seguida de un punto. Ejemplo: 4-z.1+3
function TSintaxisExpresionAlgebraica.VariableluegoPunto(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 in ['a'..'z']) then
            if (car2 = '.') then
            begin
              VariableluegoPunto := true;
              exit;
            end;
    end;

    //No encontró un número seguido de un paréntesis que abre
    VariableluegoPunto := false;
end;
```

15. Un punto seguido de una variable. Ejemplo: 7-2.p+1

El algoritmo chequea que si el carácter es un punto y lo que sigue es una letra entonces es un error.

C++

```
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
bool SintaxisExpresionAlgebraica::PuntoluegoVariable(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
		if (car1 == '.')
			if (car2 >= 'a' && car2 <= 'z') return true;
	}

	//No encontró un punto seguido de una variable
	return false;
}
```

Visual C# .NET 2010

```
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
private Boolean PuntoluegoVariable(string expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.Length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr[pos];   //Extrae un carácter
        car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == '.')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un punto seguido de una variable
    return false;
}
```

Visual Basic .NET 2010

```
'15.    Un punto seguido de una variable. Ejemplo: 7-2.p+1
Private Function PuntoluegoVariable(ByVal expr As String) As Boolean
    Dim largo, pos As Integer
    Dim car1, car2 As Char

    largo = expr.Length

    'Va de carácter en carácter
    For pos = 0 To largo - 2 Step 1
        car1 = expr(pos)     'Extrae un carácter
        car2 = expr(pos + 1) 'Extrae el siguiente carácter

        'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        If car1 = "." Then
            If car2 >= "a" And car2 <= "z" Then Return True
        End If
    Next

    'No encontró un punto seguido de una variable
    Return False
End Function
```

Java

```java
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
private Boolean PuntoluegoVariable(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == '.')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un punto seguido de una variable
    return false;
}
```

PHP

```php
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function PuntoluegoVariable($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos <= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if ($car1 == '.')
            if ($car2 >= 'a' && $car2 <= 'z') return true;
    }

    //No encontró un punto seguido de una variable
    return false;
}
```

JavaScript

```javascript
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function PuntoluegoVariable(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 == '.')
            if (car2 >= 'a' && car2 <= 'z') return true;
    }

    //No encontró un punto seguido de una variable
    return false;
}
```

Object Pascal

```
//15. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function TSintaxisExpresionAlgebraica.PuntoluegoVariable(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 = '.') then
            if (car2 in ['a'..'z']) then
            begin
              PuntoluegoVariable := true;
              exit;
            end;
    end;

    //No encontró un número seguido de un paréntesis que abre
    PuntoluegoVariable := false;
end;
```

16. Un número antes de una variable. Ejemplo: 3x+1

El algoritmo chequea que si el carácter es un número y lo que sigue es una letra entonces es un error. Algebraicamente es aceptable pero esto vuelve más complejo de hacer el evaluador y muy seguramente más lento. En vez de escribir "3x+1", el usuario debe usar "3*x+1"

C++

```
//16. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
bool SintaxisExpresionAlgebraica::NumeroAntesVariable(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
		if (car1 >= '0' && car1 <= '9')
			if (car2 >= 'a' && car2 <= 'z') return true;
	}

	//No encontró un número antes de una variable
	return false;
}
```

Visual C# .NET 2010

```
    //16. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
    private Boolean NumeroAntesVariable(string expr)
    {
        int largo, pos;
        char car1, car2;

        largo = expr.Length;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 2; pos++)
        {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 >= '0' && car1 <= '9')
                if (car2 >= 'a' && car2 <= 'z') return true;
        }

        //No encontró un número antes de una variable
        return false;
    }
```

Visual Basic .NET 2010

```
    '16. Un número antes de una variable. Ejemplo: 3x+1
    'Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
    Private Function NumeroAntesVariable(ByVal expr As String) As Boolean
        Dim largo, pos As Integer
        Dim car1, car2 As Char

        largo = expr.Length

        'Va de carácter en carácter
        For pos = 0 To largo - 2 Step 1
            car1 = expr(pos)     'Extrae un carácter
            car2 = expr(pos + 1) 'Extrae el siguiente carácter

            'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            If car1 >= "0" And car1 <= "9" Then
                If car2 >= "a" And car2 <= "z" Then Return True
            End If
        Next

        'No encontró un número antes de una variable
        Return False
    End Function
```

Java

```
    //16. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
    private Boolean NumeroAntesVariable(String expr)
    {
        int largo, pos;
        char car1, car2;

        largo = expr.length();

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 2; pos++)
        {
            car1 = expr.charAt(pos);   //Extrae un carácter
            car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 >= '0' && car1 <= '9')
                if (car2 >= 'a' && car2 <= 'z') return true;
        }

        //No encontró un número antes de una variable
        return false;
    }
```

PHP

```
      //16. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
    function NumeroAntesVariable($expr)
    {
        $largo = strlen($expr);

        //Va de carácter en carácter
        for($pos = 0; $pos <= $largo - 2; $pos++)
        {
            $car1 = $expr{$pos};   //Extrae un carácter
            $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if ($car1 >= '0' && $car1 <= '9')
                if ($car2 >= 'a' && $car2 <= 'z') return true;
        }

        //No encontró un número antes de una variable
        return false;
    }
```

JavaScript

```
    //16. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
    function NumeroAntesVariable(expr)
    {
        var largo, pos;
        var car1, car2;

        largo = expr.length;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 2; pos++)
        {
            car1 = expr.charAt(pos);   //Extrae un carácter
            car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 >= '0' && car1 <= '9')
                if (car2 >= 'a' && car2 <= 'z') return true;
        }

        //No encontró un número antes de una variable
        return false;
    }
```

Object Pascal

```
    //16. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    function TSintaxisExpresionAlgebraica.NumeroAntesVariable(expr: string): boolean;
    var
        largo, pos: integer;
        car1, car2: char;
    begin
        largo := length(expr);

        //Va de carácter en carácter
        for pos := 1 to largo - 1 do
        begin
            car1 := expr[pos];   //Extrae un carácter
            car2 := expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 in ['0'..'9']) then
                if (car2 in ['a'..'z']) then
                begin
                  NumeroAntesVariable := true;
                  exit;
                end;
        end;

        //No encontró un número seguido de un paréntesis que abre
        NumeroAntesVariable := false;
    end;
```

17. Un número después de una variable. Ejemplo: x21+4

El algoritmo chequea que si el carácter es una letra y lo que sigue es un número entonces es un error.

C++

```
//17. Un número después de una variable. Ejemplo: x21+4
bool SintaxisExpresionAlgebraica::VariableDespuesNumero(char *expr)
{
	int pos;
	char car1, car2;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos+1); pos++)
	{
		car1 = expr[pos];   //Extrae un carácter
		car2 = expr[pos+1]; //Extrae el siguiente carácter

		//Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
		if (car1 >= 'a' && car1 <= 'z')
			if (car2 >= '0' && car2 <= '9') return true;
	}

	//No encontró un número después de una variable
	return false;
}
```

Visual C# .NET 2010

```
    //17. Un número después de una variable. Ejemplo: x21+4
    private Boolean VariableDespuesNumero(string expr)
    {
        int largo, pos;
        char car1, car2;

        largo = expr.Length;

        //Va de carácter en carácter
        for(pos = 0; pos<= largo - 2; pos++)
        {
            car1 = expr[pos];   //Extrae un carácter
            car2 = expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (car1 >= 'a' && car1 <= 'z')
                if (car2 >= '0' && car2 <= '9') return true;
        }

        //No encontró un número después de una variable
        return false;
    }
```

Visual Basic .NET 2010

```
    '17. Un número después de una variable. Ejemplo: x21+4
    Private Function VariableDespuesNumero(ByVal expr As String) As Boolean
        Dim largo, pos As Integer
        Dim car1, car2 As Char

        largo = expr.Length

        'Va de carácter en carácter
        For pos = 0 To largo - 2 Step 1
            car1 = expr(pos)      'Extrae un carácter
            car2 = expr(pos + 1) 'Extrae el siguiente carácter

            'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            If car1 >= "a" And car1 <= "z" Then
                If car2 >= "0" And car2 <= "9" Then Return True
            End If
        Next

        'No encontró un número después de una variable
        Return False
    End Function
```

Java

```java
//17. Un número después de una variable. Ejemplo: x21+4
private Boolean VariableDespuesNumero(String expr)
{
    int largo, pos;
    char car1, car2;

    largo = expr.length();

    //Va de carácter en carácter
    for(pos = 0; pos<= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 >= 'a' && car1 <= 'z')
            if (car2 >= '0' && car2 <= '9') return true;
    }

    //No encontró un número después de una variable
    return false;
}
```

PHP

```php
//17. Un número después de una variable. Ejemplo: x21+4
function VariableDespuesNumero($expr)
{
    $largo = strlen($expr);

    //Va de carácter en carácter
    for($pos = 0; $pos<= $largo - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if ($car1 >= 'a' && $car1 <= 'z')
            if ($car2 >= '0' && $car2 <= '9') return true;
    }

    //No encontró un número después de una variable
    return false;
}
```

JavaScript

```javascript
//17. Un número después de una variable. Ejemplo: x21+4
function VariableDespuesNumero(expr)
{
    var largo, pos;
    var car1, car2;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos<= largo - 2; pos++)
    {
        car1 = expr.charAt(pos);   //Extrae un carácter
        car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 >= 'a' && car1 <= 'z')
            if (car2 >= '0' && car2 <= '9') return true;
    }

    //No encontró un número después de una variable
    return false;
}
```

Object Pascal

```
//17. Un número después de una variable. Ejemplo: x21+4
function TSintaxisExpresionAlgebraica.VariableDespuesNumero(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2: char;
begin
    largo := length(expr);

    //Va de carácter en carácter
    for pos := 1 to largo - 1 do
    begin
        car1 := expr[pos];   //Extrae un carácter
        car2 := expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (car1 in ['a'..'z']) then
            if (car2 in ['0'..'9']) then
            begin
              VariableDespuesNumero := true;
              exit;
            end;
    end;

    //No encontró un número después de una variable
    VariableDespuesNumero := false;
end;
```

18. Haya caracteres inválidos. Ejemplo: 4+@-1

El algoritmo chequea si hay caracteres no permitidos dentro de la expresión algebraica.

C++

```
//18. Haya caracteres inválidos. Ejemplo: 4+@-1
bool SintaxisExpresionAlgebraica::CaracterInvalido(char *expr)
{
	char *caractervalido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
	int tamcaractervalido;

	int pos, valida;
	char car1, car2;
	bool invalidocaracter;

	//Va de carácter en carácter
	for(pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];

		//Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
		invalidocaracter = true;
		for(valida = 0; *(caractervalido+valida); valida++)
		{
			car2 = caractervalido[valida];
			if (car1 == car2) invalidocaracter = false;
		}
		if (invalidocaracter ) return true;
	}
	return false;
}
```

Visual C# .NET 2010

```
    //18. Haya caracteres inválidos. Ejemplo: 4+@-1
    private Boolean CaracterInvalido(string expr)
    {
        string caractervalido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
        int tamcaractervalido;
        tamcaractervalido = caractervalido.Length;

        int largo, pos, valida;
        char car1, car2;
        Boolean invalidocaracter;

        largo = expr.Length;

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 1; pos++)
        {
            car1 = expr[pos];

            //Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
            invalidocaracter = true;
            for(valida = 0; valida <= tamcaractervalido - 1; valida++)
            {
                car2 = caractervalido[valida];
                if (car1 == car2) invalidocaracter = false;
            }
            if (invalidocaracter ) return true;
        }
        return false;
    }
```

Visual Basic .NET 2010

```
    '18. Haya caracteres inválidos.  Ejemplo: 4+@-1
    Private Function CaracterInvalido(ByVal expr As String) As Boolean
        Dim caractervalido As String = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()"
        Dim tamcaractervalido As Integer
        tamcaractervalido = caractervalido.Length

        Dim largo, pos, valida As Integer
        Dim car1, car2 As Char
        Dim invalidocaracter As Boolean

        largo = expr.Length

        'Va de carácter en carácter
        For pos = 0 To largo - 1 Step 1
            car1 = expr(pos)

            'Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
            invalidocaracter = True
            For valida = 0 To tamcaractervalido - 1 Step 1
                car2 = caractervalido(valida)
                If car1 = car2 Then invalidocaracter = False
            Next
            If invalidocaracter = True Then Return True
        Next
        Return False
    End Function
```

Java

```
    //18. Haya caracteres inválidos. Ejemplo: 4+@-1
    private Boolean CaracterInvalido(String expr)
    {
        String caractervalido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
        int tamcaractervalido;
        tamcaractervalido = caractervalido.length();

        int largo, pos, valida;
        char car1, car2;
        Boolean invalidocaracter;

        largo = expr.length();

        //Va de carácter en carácter
        for(pos = 0; pos <= largo - 1; pos++)
        {
            car1 = expr.charAt(pos);

            //Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
            invalidocaracter = true;
            for(valida = 0; valida <= tamcaractervalido - 1; valida++)
            {
                car2 = caractervalido.charAt(valida);
                if (car1 == car2) invalidocaracter = false;
            }
            if (invalidocaracter ) return true;
        }
        return false;
    }
```

PHP

```
    //18. Haya caracteres inválidos. Ejemplo: 4+@-1
    function CaracterInvalido($expr)
    {
        $caractervalido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
            $tamcaractervalido = strlen($caractervalido);

        $largo = strlen($expr);

        //Va de carácter en carácter
        for($pos = 0; $pos <= $largo - 1; $pos++)
        {
            $car1 = $expr{$pos};

            //Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
            $invalidocaracter = true;
            for($valida = 0; $valida <= $tamcaractervalido - 1; $valida++)
            {
                $car2 = $caractervalido{$valida};
                if ($car1 == $car2) $invalidocaracter = false;
            }
            if ($invalidocaracter) return true;
        }
        return false;
    }
```

JavaScript

```
//18. Haya caracteres inválidos. Ejemplo: 4+@-1
function CaracterInvalido(expr)
{
    var caractervalido = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    var tamcaractervalido;
    tamcaractervalido = caractervalido.length;

    var largo, pos, valida;
    var car1, car2;
    var invalidocaracter;

    largo = expr.length;

    //Va de carácter en carácter
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);

        //Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
        invalidocaracter = true;
        for(valida = 0; valida <= tamcaractervalido - 1; valida++)
        {
            car2 = caractervalido.charAt(valida);
            if (car1 == car2) invalidocaracter = false;
        }
        if (invalidocaracter ) return true;
    }
    return false;
}
```

Object Pascal

```
//18. Haya caracteres inválidos. Ejemplo: 4+@-1
function TSintaxisExpresionAlgebraica.CaracterInvalido(expr: string): boolean;
var
    caractervalido: string;
    largo, pos, valida, tamcaractervalido: integer;
    car1, car2: char;
    invalidocaracter: boolean;
begin
    largo := length(expr);
    caractervalido := 'abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()';
    tamcaractervalido := length(caractervalido);

    //Va de carácter en carácter
    for pos := 1 to largo do
    begin
        car1 := expr[pos];

        //Por defecto el carácter es inválido hasta que es encontrado en la lista de los válidos
        invalidocaracter := true;
        for valida := 1 to tamcaractervalido do
        begin
            car2 := caractervalido[valida];
            if (car1 = car2) then invalidocaracter := false;
        end;
        if (invalidocaracter) then
        begin
          CaracterInvalido := true;
          exit;
        end;
    end;
    CaracterInvalido := false;
end;
```

19. Chequea si hay 4 o más variables seguidas

El algoritmo chequea si hay cuatro letras seguidas, dado el caso, es un error.

C++

```
//19. Chequea si hay 4 o más variables seguidas
bool SintaxisExpresionAlgebraica::Chequea4letras(char *expr)
{
	int pos;

	char car1, car2, car3, car4;

	for(pos = 0; *(expr+pos+3); pos++)
	{
		car1 = expr[pos];
		car2 = expr[pos+1];
		car3 = expr[pos+2];
		car4 = expr[pos+3];

		if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
			return true;
	}
	return false;
}
```

Visual C# .NET 2010

```
    //19. Chequea si hay 4 o más variables seguidas
    private Boolean Chequea4letras(string expr)
    {
        int largo, pos;
        largo = expr.Length;

        char car1, car2, car3, car4;

        for(pos = 0; pos <= largo - 4; pos++)
        {
            car1 = expr[pos];
            car2 = expr[pos + 1];
            car3 = expr[pos + 2];
            car4 = expr[pos + 3];

            if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
                return true;
        }
        return false;
    }
```

Visual Basic .NET 2010

```
    '19. Chequea si hay 4 o más variables seguidas
    Private Function Chequea4letras(ByVal expr As String) As Boolean
        Dim largo, pos As Integer
        largo = expr.Length

        Dim car1, car2, car3, car4 As Char

        For pos = 0 To largo - 4 Step 1
            car1 = expr(pos)
            car2 = expr(pos + 1)
            car3 = expr(pos + 2)
            car4 = expr(pos + 3)

            If car1 >= "a" And car1 <= "z" And car2 >= "a" And car2 <= "z" And car3 >= "a" And car3 <= "z" And car4 >= "a" And car4 <= "z" Then
                Return True
            End If
        Next
        Return False

    End Function
```

Java

```java
    //19. Chequea si hay 4 o más variables seguidas
    private Boolean Chequea4letras(String expr)
    {
        int largo, pos;
        largo = expr.length();

        char car1, car2, car3, car4;

        for(pos = 0; pos <= largo - 4; pos++)
        {
            car1 = expr.charAt(pos);
            car2 = expr.charAt(pos + 1);
            car3 = expr.charAt(pos + 2);
            car4 = expr.charAt(pos + 3);

            if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <=
'z')
                return true;
        }
        return false;
    }
```

PHP

```php
    //19. Chequea si hay 4 o más variables seguidas
    function Chequea4letras($expr)
    {
        $largo = strlen($expr);

        for($pos = 0; $pos <= $largo - 4; $pos++)
        {
            $car1 = $expr{$pos};
            $car2 = $expr{$pos + 1};
            $car3 = $expr{$pos + 2};
            $car4 = $expr{$pos + 3};

            if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z' && $car4 >= 'a' &&
$car4 <= 'z')
                return true;
        }
        return false;
    }
```

JavaScript

```javascript
    //19. Chequea si hay 4 o más variables seguidas
    function Chequea4letras(expr)
    {
        var largo, pos;
        largo = expr.length;

        var car1, car2, car3, car4;

        for(pos = 0; pos <= largo - 4; pos++)
        {
            car1 = expr.charAt(pos);
            car2 = expr.charAt(pos + 1);
            car3 = expr.charAt(pos + 2);
            car4 = expr.charAt(pos + 3);

            if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <=
'z')
                return true;
        }
        return false;
    }
```

Object Pascal

```
//19. Chequea si hay 4 o más variables seguidas
function TSintaxisExpresionAlgebraica.Chequea4letras(expr: string): boolean;
var
    largo, pos: integer;
    car1, car2, car3, car4: char;
begin
    largo := length(expr);

    for pos := 1 to largo - 3 do
    begin
        car1 := expr[pos];
        car2 := expr[pos + 1];
        car3 := expr[pos + 2];
        car4 := expr[pos + 3];

        if (car1 in ['a'..'z']) and (car2 in ['a'..'z']) and (car3 in ['a'..'z']) and (car4 in ['a'..'z']) then
        begin
            Chequea4letras := true;
            exit;
        end;
    end;
    Chequea4letras := false;
end;
```

20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no

Si el algoritmo encuentra tres letras seguidas, entonces verifica si sigue un paréntesis que abre, dado el caso entonces chequea si esas tres letras conforman el nombre de una función aceptada por el evaluador de expresiones.

C++

```
//20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
bool SintaxisExpresionAlgebraica::FuncionInvalida(char *expr)
{
	bool chequeafuncion;
	char car1, car2, car3, car4;
	int largo, pos;

	largo = strlen(expr);

	for(pos = 0; *(expr+pos+2); pos++)
	{
		car1 = expr[pos];
		car2 = expr[pos+1];
		car3 = expr[pos+2];

		//Si encuentra tres letras seguidas
		if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
		{
			if (pos < largo - 4)
			{
				car4 = *(expr+pos+3);

				//Si el siguiente carácter es (
				if (car4 == '(')
				{
					chequeafuncion = EsUnaFuncion(car1, car2, car3);
					if (chequeafuncion == false) return true;
				}
				else
					return true; //Hay un error porque no hay paréntesis
			}
			else
				return true; //Hay un error porque no sigue paréntesis
		}
	}
	return false;
}
```

Visual C# .NET 2010

```
    //20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
    private Boolean FuncionInvalida(string expr)
    {
        Boolean chequeafuncion;
        char car1, car2, car3, car4;
        int largo, pos;
        largo = expr.Length;

        for(pos = 0; pos <= largo - 3; pos++)
        {
            car1 = expr[pos];
            car2 = expr[pos + 1];
            car3 = expr[pos + 2];

            //Si encuentra tres letras seguidas
            if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
            {
                if (pos < largo - 4)
                {
                    car4 = expr[pos + 3];

                    //Si el siguiente carácter es (
                    if (car4 == '(')
                    {
                        chequeafuncion = EsUnaFuncion(car1, car2, car3);
                        if (chequeafuncion == false) return true;
                    }
                    else
                        return true; //Hay un error porque no hay paréntesis
                }
                else
                    return true; //Hay un error porque no sigue paréntesis
            }
        }
        return false;
    }

    //Chequea si las tres letras enviadas son una función
    private Boolean EsUnaFuncion(char car1, char car2, char car3)
    {
        string listafunciones;
        char listfunc1, listfunc2, listfunc3;
        int tamfunciones, pos;

        listafunciones = "sinsencostanabslogexpsqrrcb";
        tamfunciones = listafunciones.Length;

        for(pos = 0; pos <= tamfunciones - 3; pos+=3)
        {
            listfunc1 = listafunciones[pos];
            listfunc2 = listafunciones[pos + 1];
            listfunc3 = listafunciones[pos + 2];

            if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return true;
        }
        return false;
    }
```

Visual Basic .NET 2010

```
    '20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
    Private Function FuncionInvalida(ByVal expr As String) As Boolean
        Dim chequeafuncion As Boolean
        Dim car1, car2, car3, car4 As Char
        Dim largo, pos As Integer
        largo = expr.Length

        For pos = 0 To largo - 3 Step 1
            car1 = expr(pos)
            car2 = expr(pos + 1)
            car3 = expr(pos + 2)

            'Si encuentra tres letras seguidas
            If car1 >= "a" And car1 <= "z" And car2 >= "a" And car2 <= "z" And car3 >= "a" And car3 <= "z" Then
                If pos < largo - 4 Then
                    car4 = expr(pos + 3)

                    'Si el siguiente carácter es (
                    If car4 = "(" Then

                        'Llama a una función que retorna si esas tres letras corresponden a una función
                        chequeafuncion = EsUnaFuncion(car1, car2, car3)
                        If chequeafuncion = False Then Return True
                    Else
                        Return True 'Hay un error porque no hay paréntesis
                    End If
                Else
                    Return True 'Hay un error porque no sigue paréntesis
                End If
            End If
        Next
        Return False
    End Function

      'Chequea si las tres letras enviadas son una función
    Private Function EsUnaFuncion(ByVal car1 As Char, ByVal car2 As Char, ByVal car3 As Char) As Boolean
        Dim listafunciones As String
        Dim listfunc1, listfunc2, listfunc3 As Char
        Dim tamfunciones, pos As Integer

        listafunciones = "sinsencostanabslogexpsqrrcb"
        tamfunciones = listafunciones.Length

        For pos = 0 To tamfunciones - 3 Step 3
            listfunc1 = listafunciones(pos)
            listfunc2 = listafunciones(pos + 1)
            listfunc3 = listafunciones(pos + 2)

            If car1 = listfunc1 And car2 = listfunc2 And car3 = listfunc3 Then Return True
        Next
        Return False

    End Function
```

Java

```
    //20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
    private Boolean FuncionInvalida(String expr)
    {
        Boolean chequeafuncion;
        char car1, car2, car3, car4;
        int largo, pos;
        largo = expr.length();

        for(pos = 0; pos <= largo - 3; pos++)
        {
            car1 = expr.charAt(pos);
            car2 = expr.charAt(pos + 1);
            car3 = expr.charAt(pos + 2);

            //Si encuentra tres letras seguidas
            if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
            {
                if (pos < largo - 4)
                {
                    car4 = expr.charAt(pos + 3);

                    //Si el siguiente carácter es (
                    if (car4 == '(')
                    {
                        chequeafuncion = EsUnaFuncion(car1, car2, car3);
                        if (chequeafuncion == false) return true;
                    }
                    else
                        return true; //Hay un error porque no hay paréntesis
                }
                else
                    return true; //Hay un error porque no sigue paréntesis
            }
        }
        return false;
    }

    //Chequea si las tres letras enviadas son una función
    private Boolean EsUnaFuncion(char car1, char car2, char car3)
    {
        String listafunciones;
        char listfunc1, listfunc2, listfunc3;
        int tamfunciones, pos;

        listafunciones = "sinsencostanabslogexpsqrrcb";
        tamfunciones = listafunciones.length();

        for(pos = 0; pos <= tamfunciones - 3; pos+=3)
        {
            listfunc1 = listafunciones.charAt(pos);
            listfunc2 = listafunciones.charAt(pos + 1);
            listfunc3 = listafunciones.charAt(pos + 2);

            if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return true;
        }
        return false;
    }
```

PHP

```php
//20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function FuncionInvalida($expr)
{
    $largo = strlen($expr);

    for($pos = 0; $pos <= $largo - 3; $pos++)
    {
        $car1 = $expr{$pos};
        $car2 = $expr{$pos + 1};
        $car3 = $expr{$pos + 2};

        //Si encuentra tres letras seguidas
        if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z')
        {
            if ($pos < $largo - 4)
            {
                $car4 = $expr{$pos + 3};

                //Si el siguiente carácter es (
                if ($car4 == '(')
                {
                    $chequeafuncion = $this->EsUnaFuncion($car1, $car2, $car3);
                    if ($chequeafuncion == false) return true;
                }
                else
                    return true; //Hay un error porque no hay paréntesis
            }
            else
                return true; //Hay un error porque no sigue paréntesis
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
function EsUnaFuncion($car1, $car2, $car3)
{
    $listafunciones = "sinsencostanabslogexpsqrrcb";
    $tamfunciones = strlen($listafunciones);

    for($pos = 0; $pos <= $tamfunciones - 3; $pos+=3)
    {
        $listfunc1 = $listafunciones{$pos};
        $listfunc2 = $listafunciones{$pos + 1};
        $listfunc3 = $listafunciones{$pos + 2};

        if ($car1 == $listfunc1 && $car2 == $listfunc2 && $car3 == $listfunc3) return true;
    }
    return false;
}
```

JavaScript

```
//20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function FuncionInvalida(expr)
{
    var chequeafuncion;
    var car1, car2, car3, car4;
    var largo, pos;
    largo = expr.length;

    for(pos = 0; pos <= largo - 3; pos++)
    {
        car1 = expr.charAt(pos);
        car2 = expr.charAt(pos + 1);
        car3 = expr.charAt(pos + 2);

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
        {
            if (pos < largo - 4)
            {
                car4 = expr.charAt(pos + 3);

                //Si el siguiente carácter es (
                if (car4 == '(')
                {
                    chequeafuncion = EsUnaFuncion(car1, car2, car3);
                    if (chequeafuncion == false) return true;
                }
                else
                    return true; //Hay un error porque no hay paréntesis
            }
            else
                return true; //Hay un error porque no sigue paréntesis
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
function EsUnaFuncion(car1, car2, car3)
{
    var listafunciones;
    var listfunc1, listfunc2, listfunc3;
    var tamfunciones, pos;

    listafunciones = "sinsencostanabslogexpsqrrcb";
    tamfunciones = listafunciones.length;

    for(pos = 0; pos <= tamfunciones - 3; pos+=3)
    {
        listfunc1 = listafunciones.charAt(pos);
        listfunc2 = listafunciones.charAt(pos + 1);
        listfunc3 = listafunciones.charAt(pos + 2);

        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return true;
    }
    return false;
}
```

Object Pascal

```
//20. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function TSintaxisExpresionAlgebraica.FuncionInvalida(expr: string): boolean;
var
    chequeafuncion: boolean;
    car1, car2, car3, car4: char;
    largo, pos: integer;
begin
    largo := length(expr);

    for pos := 1 to largo - 2 do
    begin
        car1 := expr[pos];
        car2 := expr[pos + 1];
        car3 := expr[pos + 2];

        //Si encuentra tres letras seguidas
        if (car1 in ['a'..'z']) and (car2 in ['a'..'z']) and (car3 in ['a'..'z']) then
        begin
            if (pos < largo - 4) then
            begin
                car4 := expr[pos + 3];

                //Si el siguiente carácter es (
                if (car4 = '(') then
                begin
                    chequeafuncion := EsUnaFuncion(car1, car2, car3);
                    if (chequeafuncion = false) then
                    begin
                       FuncionInvalida := true;
                       exit;
                    end;
                end
                else
                begin
                    FuncionInvalida := true; //Hay un error porque no hay paréntesis
                    exit;
                end;
            end
            else
            begin
                FuncionInvalida := true; //Hay un error porque no sigue paréntesis
                exit;
            end
        end;
    end;
    FuncionInvalida := false;
end;

//Chequea si las tres letras enviadas son una función
function TSintaxisExpresionAlgebraica.EsUnaFuncion(car1, car2, car3: char): boolean;
var
    listafunciones: string;
    listfunc1, listfunc2, listfunc3: char;
    tamfunciones, pos: integer;
begin
    listafunciones := 'sinsencostanabslogexpsqrrcb';
    tamfunciones := length(listafunciones);
    pos := 1;
    while (pos <= tamfunciones - 2) do
    begin
        listfunc1 := listafunciones[pos];
        listfunc2 := listafunciones[pos + 1];
        listfunc3 := listafunciones[pos + 2];

        if (car1 = listfunc1) and (car2 = listfunc2) and (car3 = listfunc3) then
        begin
          EsUnaFuncion := true;
          exit;
        end;

        pos := pos + 3;
    end;
    EsUnaFuncion := false;
end;
```

21. Si detecta sólo dos letras seguidas es un error. Ejemplo: 4+tk+8.9

El algoritmo chequea que si hay sólo dos letras seguidas es un error.

C++

```
//21. Si detecta sólo dos letras seguidas es un error
bool SintaxisExpresionAlgebraica::VariableInvalida(char *expr)
{
	int pos, cuentaletras;
	char car1;

	cuentaletras = 0;
	for(pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];
		if (car1 >= 'a' && car1 <= 'z')
			cuentaletras++;
		else
		{
			if (cuentaletras == 2) return true;
			cuentaletras = 0;
		}
	}
	if (cuentaletras == 2) return true;
	return false;
}
```

Visual C# .NET 2010

```
    //21. Si detecta sólo dos letras seguidas es un error
    private Boolean VariableInvalida(string expr)
    {
        int largo, pos, cuentaletras;
        char car1;

        largo = expr.Length;

        cuentaletras = 0;
        for(pos = 0; pos <= largo - 1; pos++)
        {
            car1 = expr[pos];
            if (car1 >= 'a' && car1 <= 'z')
                cuentaletras = cuentaletras + 1;
            else
            {
                if (cuentaletras == 2) return true;
                cuentaletras = 0;
            }
        }
        if (cuentaletras == 2) return true;
        return false;
    }
```

Visual Basic .NET 2010

```
    '21. Si detecta sólo dos letras seguidas es un error
    Private Function VariableInvalida(ByVal expr As String) As Boolean
        Dim largo, pos, cuentaletras As Integer
        Dim car1 As Char

        largo = expr.Length

        cuentaletras = 0

        'Va de letra en letra
        For pos = 0 To largo - 1 Step 1
            car1 = expr(pos)
            If car1 >= "a" And car1 <= "z" Then
                cuentaletras = cuentaletras + 1
            Else
                'Si sólo contó dos letras entonces es un error
                If cuentaletras = 2 Then Return True
                cuentaletras = 0
            End If
        Next
        If cuentaletras = 2 Then Return True
        Return False
    End Function
```

Java

```java
//21. Si detecta sólo dos letras seguidas es un error
private Boolean VariableInvalida(String expr)
{
    int largo, pos, cuentaletras;
    char car1;

    largo = expr.length();

    cuentaletras = 0;
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras++;
        else
        {
            if (cuentaletras == 2) return true;
            cuentaletras = 0;
        }
    }
    if (cuentaletras == 2) return true;
    return false;
}
```

PHP

```php
//21. Si detecta sólo dos letras seguidas es un error
function VariableInvalida($expr)
{
    $largo = strlen($expr);

    $cuentaletras = 0;
    for($pos = 0; $pos <= $largo - 1; $pos++)
    {
        $car1 = $expr{$pos};
        if ($car1 >= 'a' && $car1 <= 'z')
            $cuentaletras = $cuentaletras + 1;
        else
        {
            if ($cuentaletras == 2) return true;
            $cuentaletras = 0;
        }
    }
    if ($cuentaletras == 2) return true;
    return false;
}
```

JavaScript

```javascript
//21. Si detecta sólo dos letras seguidas es un error
function VariableInvalida(expr)
{
    var largo, pos, cuentaletras;
    var car1;

    largo = expr.length;

    cuentaletras = 0;
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras = cuentaletras + 1;
        else
        {
            if (cuentaletras == 2) return true;
            cuentaletras = 0;
        }
    }
    if (cuentaletras == 2) return true;
    return false;
}
```

Object Pascal

```
//21. Si detecta sólo dos letras seguidas es un error
function TSintaxisExpresionAlgebraica.VariableInvalida(expr: string): boolean;
var
    largo, pos, cuentaletras: integer;
    car1: char;
begin
    largo := length(expr);

    cuentaletras := 0;
    for pos := 1 to largo do
    begin
        car1 := expr[pos];
        if (car1 in ['a'..'z']) then
            cuentaletras := cuentaletras + 1
        else
        begin
            if (cuentaletras = 2) then
            begin
              VariableInvalida := true;
              exit;
            end;
            cuentaletras := 0;
        end;
    end;
    if (cuentaletras = 2) then
      VariableInvalida := true
    else
      VariableInvalida := false;
end;
```

22. Antes de paréntesis que abre hay una letra. Ejemplo: 4*a(6+b)

El algoritmo chequea que si sólo hay una letra y luego sigue un paréntesis que abre entonces es un error.

C++

```
//22. Antes de paréntesis que abre hay una letra
bool SintaxisExpresionAlgebraica::VariableParentesisAbre(char *expr)
{
	int pos, cuentaletras;
	char car1;

	cuentaletras = 0;
	for(pos = 0; *(expr+pos); pos++)
	{
		car1 = expr[pos];
		if (car1 >= 'a' && car1 <= 'z')
			cuentaletras++;
		else if (car1 == '(')
			if (cuentaletras == 1) return true;
		else
			cuentaletras = 0;
	}
	return false;
}
```

Visual C# .NET 2010

```
    //22. Antes de paréntesis que abre hay una letra
    private Boolean VariableParentesisAbre(string expr)
    {
        int largo, pos, cuentaletras;
        char car1;

        largo = expr.Length;
        cuentaletras = 0;
        for(pos = 0; pos <= largo - 1; pos++)
        {
            car1 = expr[pos];
            if (car1 >= 'a' && car1 <= 'z')
                cuentaletras++;
            else if (car1 == '(')
                if (cuentaletras == 1) return true;
            else
                cuentaletras = 0;
        }
        return false;
    }
```

Visual Basic .NET 2010

```
    '22. Antes de paréntesis que abre hay una letra. Ejemplo: 4*a(6+b)
    Private Function VariableParentesisAbre(ByVal expr As String) As Boolean
        Dim largo, pos, cuentaletras As Integer
        Dim car1 As Char

        largo = expr.Length
        cuentaletras = 0
        For pos = 0 To largo - 1 Step 1
            car1 = expr(pos)
            If car1 >= "a" And car1 <= "z" Then
                cuentaletras = cuentaletras + 1
            ElseIf car1 = "(" Then
                'Si sólo contó una letra antes de paréntesis que abre entonces es un error
                If cuentaletras = 1 Then Return True
            Else
                cuentaletras = 0
            End If
        Next
        Return False
    End Function
```

Java

```java
//22. Antes de paréntesis que abre hay una letra
private Boolean VariableParentesisAbre(String expr)
{
    int largo, pos, cuentaletras;
    char car1;

    largo = expr.length();
    cuentaletras = 0;
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras++;
        else if (car1 == '(')
            if (cuentaletras == 1) return true;
        else
            cuentaletras = 0;
    }
    return false;
}
```

PHP

```php
//22. Antes de paréntesis que abre hay una letra
function VariableParentesisAbre($expr)
{
    $largo = strlen($expr);
    $cuentaletras = 0;

    for($pos = 0; $pos <= $largo - 1; $pos++)
    {
        $car1 = $expr{$pos};
        if ($car1 >= 'a' && $car1 <= 'z')
            $cuentaletras++;
        else if ($car1 == '(')
            if ($cuentaletras == 1) return true;
        else
            $cuentaletras = 0;
    }
    return false;
}
```

JavaScript

```javascript
//22. Antes de paréntesis que abre hay una letra
function VariableParentesisAbre(expr)
{
    var largo, pos, cuentaletras;
    var car1;

    largo = expr.length;
    cuentaletras = 0;
    for(pos = 0; pos <= largo - 1; pos++)
    {
        car1 = expr.charAt(pos);
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras++;
        else if (car1 == '(')
            if (cuentaletras == 1) return true;
        else
            cuentaletras = 0;
    }
    return false;
}
```

Object Pascal

```
    //22. Antes de paréntesis que abre hay una letra
    function TSintaxisExpresionAlgebraica.VariableParentesisAbre(expr: string): boolean;
    var
        largo, pos, cuentaletras: integer;
        car1: char;
    begin
        largo := length(expr);
        cuentaletras := 0;
        for pos := 0 to largo do
        begin
            car1 := expr[pos];
            if (car1 in ['a'..'z']) then
                cuentaletras := cuentaletras + 1
            else if (car1 = '(') then
                if (cuentaletras = 1) then
                begin
                  VariableParentesisAbre := true;
                  exit;
                end
            else
                cuentaletras := 0;
        end;
        VariableParentesisAbre := false;
    end;
```

Llamado de los métodos

Los métodos anteriores son privados de la clase, luego debe haber un método público que se encarga de recibir la expresión algebraica, y hacerle las validaciones en orden, si encuentra una falla entonces retorna un valor diferente de cero, en caso de encontrar una falla sintáctica en la expresión, detiene el análisis y retorna el error encontrado.

C++

```
//Valida la expresión algebraica
int SintaxisExpresionAlgebraica::EvaluaSintaxis(char *exprOrig)
{
	int tamano = strlen(exprOrig) + 1;

	//Reserva memoria para la segunda cadena
	char *expr = (char*) malloc ( tamano );

	//Inicializa la memoria reservada
	for (int cont=0; cont<tamano; cont++)
		expr[cont] ='\0';

	//Quita los espacios de la expresión original
	SinEspacios(expr, exprOrig);

	//Hace 22 pruebas de sintaxis
	if (strlen(expr) == 0) { free(expr); return 1; }
	if (DosOperadoresSeguidos(expr)) { free(expr); return 2; }
	if (OperadorParentesisCierra(expr)) { free(expr); return 3; }
	if (ParentesisAbreOperador(expr)) { free(expr); return 4; }
	if (IniciaOperador(expr)) { free(expr); return 5; }
	if (FinalizaOperador(expr)) { free(expr); return 6; }
	if (ParentesisDesbalanceados(expr)) { free(expr); return 7; }
	if (ParentesisVacio(expr)) { free(expr); return 8; }
	if (ParentesisBalanceIncorrecto(expr)) { free(expr); return 9; }
	if (ParentesisCierraNumero(expr)) { free(expr); return 10; }
	if (NumeroParentesisAbre(expr)) { free(expr); return 11; }
	if (DoblePuntoNumero(expr)) { free(expr); return 12; }
	if (ParentesisCierraVariable(expr)) { free(expr); return 13; }
	if (VariableluegoPunto(expr)) { free(expr); return 14; }
	if (PuntoluegoVariable(expr)) { free(expr); return 15; }
	if (NumeroAntesVariable(expr)) { free(expr); return 16; }
	if (VariableDespuesNumero(expr)) { free(expr); return 17; }
	if (CaracterInvalido(expr)) { free(expr); return 18; }
	if (Chequea4letras(expr)) { free(expr); return 19; }
	if (FuncionInvalida(expr)) { free(expr); return 20; }
	if (VariableInvalida(expr)) { free(expr); return 21; }
	if (VariableParentesisAbre(expr)) { free(expr); return 22; }

	//No se detectó error de sintaxis
	free(expr);
	return 0;
}
```

Visual C# .NET 2010

```
    //Valida la expresión algebraica
    public int EvaluaSintaxis(string exprOrig)
    {
        //Quita los espacios de la expresión original
        string expr;
        expr = SinEspacios(exprOrig);

        //Hace 22 pruebas de sintaxis
        if (expr.Length == 0) return 1;
        if (DosOperadoresSeguidos(expr)) return 2;
        if (OperadorParentesisCierra(expr)) return 3;
        if (ParentesisAbreOperador(expr)) return 4;
        if (IniciaOperador(expr)) return 5;
        if (FinalizaOperador(expr)) return 6;
        if (ParentesisDesbalanceados(expr)) return 7;
        if (ParentesisVacio(expr)) return 8;
        if (ParentesisBalanceIncorrecto(expr)) return 9;
        if (ParentesisCierraNumero(expr)) return 10;
        if (NumeroParentesisAbre(expr)) return 11;
        if (DoblePuntoNumero(expr)) return 12;
        if (ParentesisCierraVariable(expr)) return 13;
        if (VariableluegoPunto(expr)) return 14;
        if (PuntoluegoVariable(expr)) return 15;
        if (NumeroAntesVariable(expr)) return 16;
        if (VariableDespuesNumero(expr)) return 17;
        if (CaracterInvalido(expr)) return 18;
        if (Chequea4letras(expr)) return 19;
        if (FuncionInvalida(expr)) return 20;
        if (VariableInvalida(expr)) return 21;
        if (VariableParentesisAbre(expr)) return 22;

        //No se detectó error de sintaxis
        return 0;
    }
```

Visual Basic .NET 2010

```
    'Valida la expresión algebraica
    Public Function EvaluaSintaxis(ByVal exprOrig As String) As Integer

        'Quita los espacios de la expresión original
        Dim expr As String
        expr = SinEspacios(exprOrig)

        'Hace 22 pruebas de sintaxis
        If expr.Length = 0 Then Return 1
        If DosOperadoresSeguidos(expr) Then Return 2
        If OperadorParentesisCierra(expr) Then Return 3
        If ParentesisAbreOperador(expr) Then Return 4
        If IniciaOperador(expr) Then Return 5
        If FinalizaOperador(expr) Then Return 6
        If ParentesisDesbalanceados(expr) Then Return 7
        If ParentesisVacio(expr) Then Return 8
        If ParentesisBalanceIncorrecto(expr) Then Return 9
        If ParentesisCierraNumero(expr) Then Return 10
        If NumeroParentesisAbre(expr) Then Return 11
        If DoblePuntoNumero(expr) Then Return 12
        If ParentesisCierraVariable(expr) Then Return 13
        If VariableluegoPunto(expr) Then Return 14
        If PuntoluegoVariable(expr) Then Return 15
        If NumeroAntesVariable(expr) Then Return 16
        If VariableDespuesNumero(expr) Then Return 17
        If CaracterInvalido(expr) Then Return 18
        If Chequea4letras(expr) Then Return 19
        If FuncionInvalida(expr) Then Return 20
        If VariableInvalida(expr) Then Return 21
        If VariableParentesisAbre(expr) Then Return 22

        'No se detectó error de sintaxis
        Return 0
    End Function
```

Java

```
    //Valida la expresión algebraica
    public int EvaluaSintaxis(String exprOrig)
    {
        //Quita los espacios de la expresión original
        String expr;
        expr = SinEspacios(exprOrig);

        //Hace 22 pruebas de sintaxis
        if (expr.length() == 0) return 1;
        if (DosOperadoresSeguidos(expr)) return 2;
        if (OperadorParentesisCierra(expr)) return 3;
        if (ParentesisAbreOperador(expr)) return 4;
        if (IniciaOperador(expr)) return 5;
        if (FinalizaOperador(expr)) return 6;
        if (ParentesisDesbalanceados(expr)) return 7;
        if (ParentesisVacio(expr)) return 8;
        if (ParentesisBalanceIncorrecto(expr)) return 9;
        if (ParentesisCierraNumero(expr)) return 10;
        if (NumeroParentesisAbre(expr)) return 11;
        if (DoblePuntoNumero(expr)) return 12;
        if (ParentesisCierraVariable(expr)) return 13;
        if (VariableluegoPunto(expr)) return 14;
        if (PuntoluegoVariable(expr)) return 15;
        if (NumeroAntesVariable(expr)) return 16;
        if (VariableDespuesNumero(expr)) return 17;
        if (CaracterInvalido(expr)) return 18;
        if (Chequea4letras(expr)) return 19;
        if (FuncionInvalida(expr)) return 20;
        if (VariableInvalida(expr)) return 21;
        if (VariableParentesisAbre(expr)) return 22;

        //No se detectó error de sintaxis
        return 0;
    }
```

PHP

```php
//Valida la expresión algebraica
public function EvaluaSintaxis($exprOrig)
{
        //Quita los espacios de la expresión original
    $expr = $this->SinEspacios($exprOrig);

    //Hace 22 pruebas de sintaxis
    if (strlen($expr) == 0) return 1;
    if ($this->DosOperadoresSeguidos($expr)) return 2;
    if ($this->OperadorParentesisCierra($expr)) return 3;
    if ($this->ParentesisAbreOperador($expr)) return 4;
    if ($this->IniciaOperador($expr)) return 5;
    if ($this->FinalizaOperador($expr)) return 6;
    if ($this->ParentesisDesbalanceados($expr)) return 7;
    if ($this->ParentesisVacio($expr)) return 8;
    if ($this->ParentesisBalanceIncorrecto($expr)) return 9;
    if ($this->ParentesisCierraNumero($expr)) return 10;
    if ($this->NumeroParentesisAbre($expr)) return 11;
    if ($this->DoblePuntoNumero($expr)) return 12;
    if ($this->ParentesisCierraVariable($expr)) return 13;
    if ($this->VariableluegoPunto($expr)) return 14;
    if ($this->PuntoluegoVariable($expr)) return 15;
    if ($this->NumeroAntesVariable($expr)) return 16;
    if ($this->VariableDespuesNumero($expr)) return 17;
    if ($this->CaracterInvalido($expr)) return 18;
    if ($this->Chequea4letras($expr)) return 19;
    if ($this->FuncionInvalida($expr)) return 20;
    if ($this->VariableInvalida($expr)) return 21;
    if ($this->VariableParentesisAbre($expr)) return 22;

    //No se detectó error de sintaxis
    return 0;
}
```

JavaScript

```javascript
//Valida la expresión algebraica
function EvaluaSintaxis(exprOrig)
{
    //Quita los espacios de la expresión original
    expr = SinEspacios(exprOrig);

    //Hace 22 pruebas de sintaxis
    if (expr.length == 0) return 1;
    if (DosOperadoresSeguidos(expr)) return 2;
    if (OperadorParentesisCierra(expr)) return 3;
    if (ParentesisAbreOperador(expr)) return 4;
    if (IniciaOperador(expr)) return 5;
    if (FinalizaOperador(expr)) return 6;
    if (ParentesisDesbalanceados(expr)) return 7;
    if (ParentesisVacio(expr)) return 8;
    if (ParentesisBalanceIncorrecto(expr)) return 9;
    if (ParentesisCierraNumero(expr)) return 10;
    if (NumeroParentesisAbre(expr)) return 11;
    if (DoblePuntoNumero(expr)) return 12;
    if (ParentesisCierraVariable(expr)) return 13;
    if (VariableluegoPunto(expr)) return 14;
    if (PuntoluegoVariable(expr)) return 15;
    if (NumeroAntesVariable(expr)) return 16;
    if (VariableDespuesNumero(expr)) return 17;
    if (CaracterInvalido(expr)) return 18;
    if (Chequea4letras(expr)) return 19;
    if (FuncionInvalida(expr)) return 20;
    if (VariableInvalida(expr)) return 21;
    if (VariableParentesisAbre(expr)) return 22;


    //No se detectó error de sintaxis
    return 0;
}
```

Object Pascal

```
    //Valida la expresión algebraica
    function TSintaxisExpresionAlgebraica.EvaluaSintaxis(exprOrig: string): integer;
    var
        expr: string;
    begin
        //Quita los espacios de la expresión original
        expr := SinEspacios(AnsiLowerCase(exprOrig));

        //Hace 22 pruebas de sintaxis
        if length(expr) = 0 then begin EvaluaSintaxis :=  1; exit;  end;
        if (DosOperadoresSeguidos(expr)) then begin EvaluaSintaxis :=  2; exit;  end;
        if (OperadorParentesisCierra(expr)) then begin EvaluaSintaxis :=  3; exit;  end;
        if (ParentesisAbreOperador(expr)) then begin EvaluaSintaxis :=  4; exit;  end;
        if (IniciaOperador(expr)) then begin EvaluaSintaxis :=  5; exit;  end;
        if (FinalizaOperador(expr)) then begin EvaluaSintaxis :=  6; exit;  end;
        if (ParentesisDesbalanceados(expr)) then begin EvaluaSintaxis :=  7; exit;  end;
        if (ParentesisVacio(expr)) then begin EvaluaSintaxis :=  8; exit;  end;
        if (ParentesisBalanceIncorrecto(expr)) then begin EvaluaSintaxis :=  9; exit;  end;
        if (ParentesisCierraNumero(expr)) then begin EvaluaSintaxis :=  10; exit;  end;
        if (NumeroParentesisAbre(expr)) then begin EvaluaSintaxis :=  11; exit;  end;
        if (DoblePuntoNumero(expr)) then begin EvaluaSintaxis :=  12; exit;  end;
        if (ParentesisCierraVariable(expr)) then begin EvaluaSintaxis :=  13; exit;  end;
        if (VariableluegoPunto(expr)) then begin EvaluaSintaxis :=  14; exit;  end;
        if (PuntoluegoVariable(expr)) then begin EvaluaSintaxis :=  15; exit;  end;
        if (NumeroAntesVariable(expr)) then begin EvaluaSintaxis :=  16; exit;  end;
        if (VariableDespuesNumero(expr)) then begin EvaluaSintaxis :=  17; exit;  end;
        if (CaracterInvalido(expr)) then begin EvaluaSintaxis :=  18; exit;  end;
        if (Chequea4letras(expr)) then begin EvaluaSintaxis :=  19; exit;  end;
        if (FuncionInvalida(expr)) then begin EvaluaSintaxis :=  20; exit;  end;
        if (VariableInvalida(expr)) then begin EvaluaSintaxis :=  21; exit;  end;
        if (VariableParentesisAbre(expr)) then begin EvaluaSintaxis :=  22; exit;  end;

        //No se detectó error de sintaxis
        EvaluaSintaxis :=  0;
    end;
```

Interpretación del diagnóstico

Una vez que se obtiene el valor entero que dice que sucedió con el chequeo de sintaxis algebraica, entonces el siguiente método retorna una frase que interpreta el resultado entero.

C++

```
//Muestra mensaje de error sintáctico
void SintaxisExpresionAlgebraica::MensajeSintaxis(int CodigoError)
{
 strcpy(mensaje, "...");
 switch(CodigoError)
 {
  case 0:
   strcpy(mensaje,"No se detecto error sintáctico en las 22 pruebas que se hicieron."); break;
  case 1:
   strcpy(mensaje, "1. Cadena vacía"); break;
  case 2:
   strcpy(mensaje, "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"); break;
  case 3:
   strcpy(mensaje, "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"); break;
  case 4:
   strcpy(mensaje, "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"); break;
  case 5:
   strcpy(mensaje, "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"); break;
  case 6:
   strcpy(mensaje, "6. Que termine con operador. Ejemplo: 12-67* 2/3-"); break;
  case 7:
   strcpy(mensaje, "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"); break;
  case 8:
   strcpy(mensaje, "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"); break;
  case 9:
   strcpy(mensaje, "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4");
   break;
  case 10:
   strcpy(mensaje, "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"); break;
  case 11:
   strcpy(mensaje, "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"); break;
  case 12:
   strcpy(mensaje, "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2"); break;
  case 13:
   strcpy(mensaje, "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"); break;
  case 14:
   strcpy(mensaje, "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"); break;
  case 15:
   strcpy(mensaje, "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"); break;
  case 16:
   strcpy(mensaje, "16. Un número antes de una variable. Ejemplo: 3x+1"); break;
  case 17:
   strcpy(mensaje, "17. Un número después de una variable. Ejemplo: x21+4"); break;
  case 18:
   strcpy(mensaje, "18. Haya caracteres inválidos. Ejemplo: 4+@-1"); break;
  case 19:
   strcpy(mensaje, "19. Hay 4 o más variables seguidas"); break;
  case 20:
   strcpy(mensaje, "20. Función inexistente"); break;
  case 21:
   strcpy(mensaje, "21. Variable invalida (solo pueden tener una letra)"); break;
  case 22:
   strcpy(mensaje, "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"); break;
 }
}
```

Visual C# .NET 2010

```
//Muestra mensaje de error sintáctico
public string MensajeSintaxis(int CodigoError)
{
 string mensaje;

 mensaje = "...";
 switch(CodigoError)
 {
  case 0:
   mensaje = "No se detecto error sintáctico en las 22 pruebas que se hicieron."; break;
  case 1:
   mensaje = "1. Cadena vacía"; break;
  case 2:
   mensaje = "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"; break;
  case 3:
   mensaje = "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"; break;
  case 4:
   mensaje = "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"; break;
  case 5:
   mensaje = "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"; break;
  case 6:
   mensaje = "6. Que termine con operador. Ejemplo: 12-67* 2/3-"; break;
  case 7:
   mensaje = "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"; break;
  case 8:
   mensaje = "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"; break;
  case 9:
   mensaje = "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4"; break;
  case 10:
   mensaje = "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"; break;
  case 11:
   mensaje = "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"; break;
  case 12:
   mensaje = "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2"; break;
  case 13:
   mensaje = "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"; break;
  case 14:
   mensaje = "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"; break;
  case 15:
   mensaje = "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"; break;
  case 16:
   mensaje = "16. Un número antes de una variable. Ejemplo: 3x+1"; break;
  case 17:
   mensaje = "17. Un número después de una variable. Ejemplo: x21+4"; break;
  case 18:
   mensaje = "18. Haya caracteres inválidos. Ejemplo: 4+@-1"; break;
  case 19:
   mensaje = "19. Hay 4 o más variables seguidas"; break;
  case 20:
   mensaje = "20. Función inexistente"; break;
  case 21:
   mensaje = "21. Variable invalida (solo pueden tener una letra)"; break;
  case 22:
   mensaje = "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"; break;
 }
 return mensaje;
}
```

Visual Basic .NET 2010

```
'Muestra mensaje de error sintáctico
Public Function mensajeSintaxis(ByVal CodigoError As Integer) As String
 Dim mensaje As String

 mensaje = "..."
 Select Case CodigoError
  Case 0
   mensaje = "No se detecto error sintáctico en las 22 pruebas que se hicieron."
  Case 1
   mensaje = "Cadena vacía"
  Case 2
   mensaje = "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"
  Case 3
   mensaje = "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"
  Case 4
   mensaje = "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"
  Case 5
   mensaje = "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"
  Case 6
   mensaje = "6. Que termine con operador. Ejemplo: 12-67* 2/3-"
  Case 7
   mensaje = "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"
  Case 8
   mensaje = "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"
  Case 9
   mensaje = "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4"
  Case 10
   mensaje = "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"
  Case 11
   mensaje = "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"
  Case 12
   mensaje = "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2"
  Case 13
   mensaje = "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"
  Case 14
   mensaje = "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"
  Case 15
   mensaje = "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"
  Case 16
   mensaje = "16. Un número antes de una variable. Ejemplo: 3x+1"
  Case 17
   mensaje = "17. Un número después de una variable. Ejemplo: x21+4"
  Case 18
   mensaje = "18. Haya caracteres inválidos. Ejemplo: 4+@-1"
  Case 19
   mensaje = "19. Hay 4 o más variables seguidas"
  Case 20
   mensaje = "20. Función inexistente"
  Case 21
   mensaje = "21. Variable invalida (solo pueden tener una letra)"
  Case 22
   mensaje = "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"
 End Select
 Return mensaje
End Function
```

Java

```
//Muestra mensaje de error sintáctico
public String MensajeSintaxis(int CodigoError)
{
 String mensaje;

 mensaje = "...";
 switch(CodigoError)
 {
  case 0:
   mensaje = "No se detecto error sintáctico en las 22 pruebas que se hicieron."; break;
  case 1:
   mensaje = "1. Cadena vacía"; break;
  case 2:
   mensaje = "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"; break;
  case 3:
   mensaje = "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"; break;
  case 4:
   mensaje = "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"; break;
  case 5:
   mensaje = "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"; break;
  case 6:
   mensaje = "6. Que termine con operador. Ejemplo: 12-67* 2/3-"; break;
  case 7:
   mensaje = "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"; break;
  case 8:
   mensaje = "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"; break;
  case 9:
   mensaje = "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4"; break;
  case 10:
   mensaje = "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"; break;
  case 11:
   mensaje = "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"; break;
  case 12:
   mensaje = "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2"; break;
  case 13:
   mensaje = "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"; break;
  case 14:
   mensaje = "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"; break;
  case 15:
   mensaje = "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"; break;
  case 16:
   mensaje = "16. Un número antes de una variable. Ejemplo: 3x+1"; break;
  case 17:
   mensaje = "17. Un número después de una variable. Ejemplo: x21+4"; break;
  case 18:
   mensaje = "18. Haya caracteres inválidos. Ejemplo: 4+@-1"; break;
  case 19:
   mensaje = "19. Hay 4 o más variables seguidas"; break;
  case 20:
   mensaje = "20. Función inexistente"; break;
  case 21:
   mensaje = "21. Variable invalida (solo pueden tener una letra)"; break;
  case 22:
   mensaje = "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"; break;
 }
 return mensaje;
}
```

PHP

```php
 //Muestra mensaje de error sintáctico
 public function MensajeSintaxis($CodigoError)
 {
  $mensaje = "...";
  switch($CodigoError)
  {
   case 0:
    $mensaje = "No se detectó error sintáctico en las 22 pruebas que se hicieron."; break;
   case 1:
    $mensaje = "1. Cadena vacía"; break;
   case 2:
    $mensaje = "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"; break;
   case 3:
    $mensaje = "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"; break;
   case 4:
    $mensaje = "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"; break;
   case 5:
    $mensaje = "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"; break;
   case 6:
    $mensaje = "6. Que termine con operador. Ejemplo: 12-67* 2/3-"; break;
   case 7:
    $mensaje = "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"; break;
   case 8:
    $mensaje = "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"; break;
   case 9:
    $mensaje = "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4";
break;
   case 10:
    $mensaje = "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"; break;
   case 11:
    $mensaje = "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"; break;
   case 12:
    $mensaje = "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2"; break;
   case 13:
    $mensaje = "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"; break;
   case 14:
    $mensaje = "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"; break;
   case 15:
    $mensaje = "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"; break;
   case 16:
    $mensaje = "16. Un número antes de una variable. Ejemplo: 3x+1"; break;
   case 17:
    $mensaje = "17. Un número después de una variable. Ejemplo: x21+4"; break;
   case 18:
    $mensaje = "18. Haya caracteres inválidos. Ejemplo: 4+@-1"; break;
   case 19:
    $mensaje = "19. Hay 4 o más variables seguidas"; break;
   case 20:
    $mensaje = "20. Función inexistente"; break;
   case 21:
    $mensaje = "21. Variable invalida (solo pueden tener una letra)"; break;
   case 22:
    $mensaje = "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"; break;
  }
  return $mensaje;
 }
```

JavaScript

```
//Muestra mensaje de error sintáctico
function MensajeSintaxis(CodigoError)
{
     var mensaje;

 mensaje = "...";
 switch(CodigoError)
 {
  case 0:
   mensaje = "No se detectó error sintáctico en las 22 pruebas que se hicieron."; break;
  case 1:
   mensaje = "1. Cadena vacía"; break;
  case 2:
   mensaje = "2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"; break;
  case 3:
   mensaje = "3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"; break;
  case 4:
   mensaje = "4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"; break;
  case 5:
   mensaje = "5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4"; break;
  case 6:
   mensaje = "6. Que termine con operador. Ejemplo: 12-67* 2/3-"; break;
  case 7:
   mensaje = "7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"; break;
  case 8:
   mensaje = "8. Que haya paréntesis vacío. Ejemplo: 2-()*3"; break;
  case 9:
   mensaje = "9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4"; break;
  case 10:
   mensaje = "10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1"; break;
  case 11:
   mensaje = "11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"; break;
  case 12:
   mensaje = "12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2"; break;
  case 13:
   mensaje = "13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"; break;
  case 14:
   mensaje = "14. Una variable seguida de un punto. Ejemplo: 4-z.1+3"; break;
  case 15:
   mensaje = "15. Un punto seguido de una variable. Ejemplo: 7-2.p+1"; break;
  case 16:
   mensaje = "16. Un número antes de una variable. Ejemplo: 3x+1"; break;
  case 17:
   mensaje = "17. Un número después de una variable. Ejemplo: x21+4"; break;
  case 18:
   mensaje = "18. Haya caracteres inválidos. Ejemplo: 4+@-1"; break;
  case 19:
   mensaje = "19. Hay 4 o más variables seguidas"; break;
  case 20:
   mensaje = "20. Función inexistente"; break;
  case 21:
   mensaje = "21. Variable invalida (solo pueden tener una letra)"; break;
  case 22:
   mensaje = "22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"; break;
 }
 return mensaje;
}
```

Object Pascal

```
    //Muestra mensaje de error sintáctico
    function TSintaxisExpresionAlgebraica.MensajeSintaxis(CodigoError: integer): string;
    var
        mensaje: string;
    begin
        mensaje := '...';
        case CodigoError of
             0: begin  mensaje := 'No se detecto error sintáctico en las 22 pruebas que se hicieron.'; end;
             1: begin  mensaje := '1. Cadena vacía'; end;
             2: begin  mensaje := '2. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3'; end;
             3: begin  mensaje := '3. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7'; end;
             4: begin  mensaje := '4. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)'; end;
             5: begin  mensaje := '5. Que empiece con operador +, *, /. Ejemplo: /12-5*2 , *17-4'; end;
             6: begin  mensaje := '6. Que termine con operador. Ejemplo: 12-67* 2/3-'; end;
             7: begin  mensaje := '7. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))'; end;
             8: begin  mensaje := '8. Que haya paréntesis vacío. Ejemplo: 2-()*3'; end;
             9: begin  mensaje := '9. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra.
Ejemplo: 2+3)-2*(4'; end;
            10: begin mensaje := '10. Un paréntesis que cierra seguido de un número. Ejemplo: (12-4)7-1'; end;
            11: begin mensaje := '11. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)'; end;
            12: begin mensaje := '12. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2'; end;
            13: begin mensaje := '13. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1'; end;
            14: begin mensaje := '14. Una variable seguida de un punto. Ejemplo: 4-z.1+3'; end;
            15: begin mensaje := '15. Un punto seguido de una variable. Ejemplo: 7-2.p+1'; end;
            16: begin mensaje := '16. Un número antes de una variable. Ejemplo: 3x+1'; end;
            17: begin mensaje := '17. Un número después de una variable. Ejemplo: x21+4'; end;
            18: begin mensaje := '18. Haya caracteres inválidos. Ejemplo: 4+@-1'; end;
            19: begin mensaje := '19. Hay 4 o más variables seguidas'; end;
            20: begin mensaje := '20. Función inexistente'; end;
            21: begin mensaje := '21. Variable invalida (solo pueden tener una letra)'; end;
            22: begin mensaje := '22. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)'; end;
        end;
        MensajeSintaxis := mensaje;
    end;
```

PARTE 2. EVALUAR UNA EXPRESIÓN SIMPLE (SIN PARÉNTESIS) DE SÓLO NÚMEROS Y LOS OPERADORES BÁSICOS

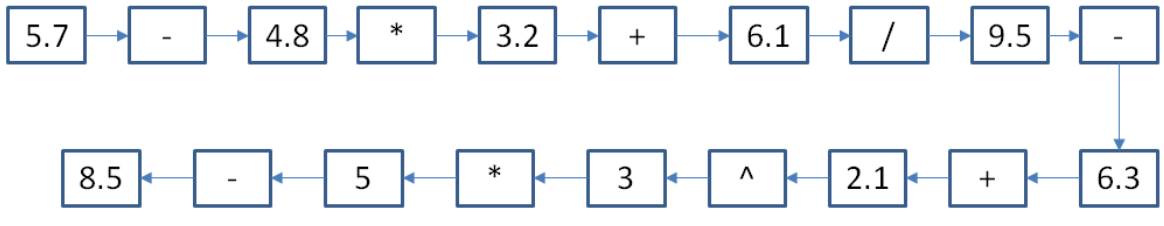
Tenemos una expresión algebraica simple que no hace uso de paréntesis y sólo tiene números reales (el separador de decimales es el punto) y los operadores básicos, como lo mostrado en la imagen:

$$5.7 - 4.8 * 3.2 + 6.1 / 9.5 - 6.3 + 2 \text{ ^3 } * 5 - 8.5$$

Se divide esta expresión en números y operadores

5.7	−	4.8	*	3.2	+	6.1	/	9.5	−	6.3	+	2	^	3	*	5	−	8.5

Cada pieza es puesta en una lista



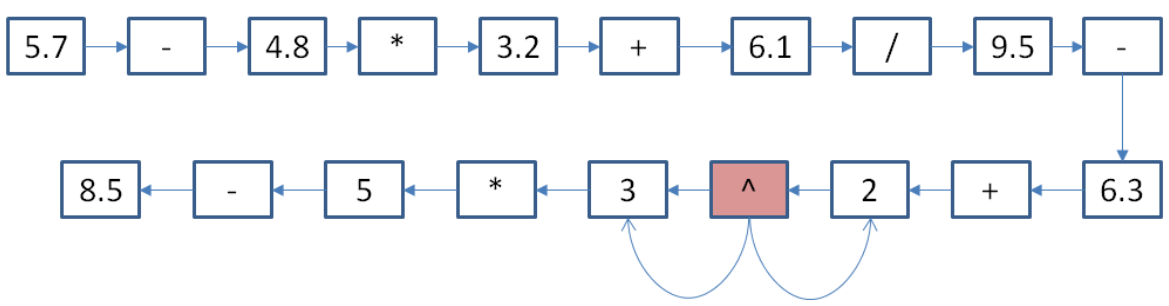
Cada pieza puede tener o un número real o un operador (+, -, *, /, ^).

El proceso de convertir la expresión algebraica a una lista se le conoce como Analizar() y es el que se implementa en los lenguajes de programación.

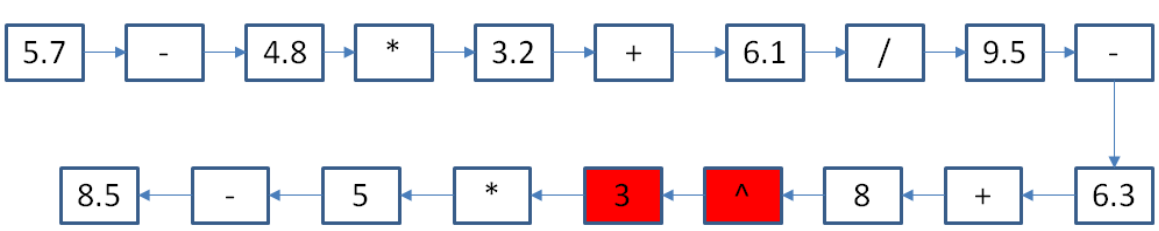
Una vez analizada la expresión, viene el proceso de Evaluar() que retorna el resultado de la expresión algebraica. El orden de evaluación es el siguiente siguiendo las reglas del álgebra:

1. Valorar la exponenciación
2. Valorar la multiplicación y la división
3. Valorar la suma y la resta

El algoritmo busca en la lista primero el operador exponenciación(^), una vez hallado busca el número anterior y el número siguiente a su posición. En el ejemplo encuentra a 2^3 que debe dar 8.

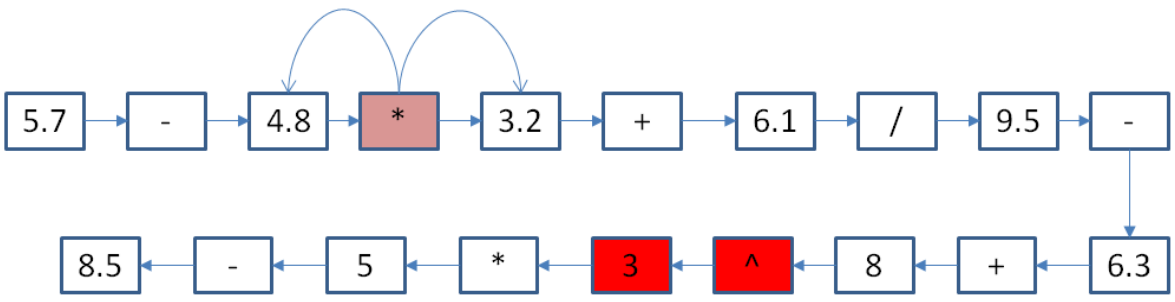


Evalúa la exponenciación

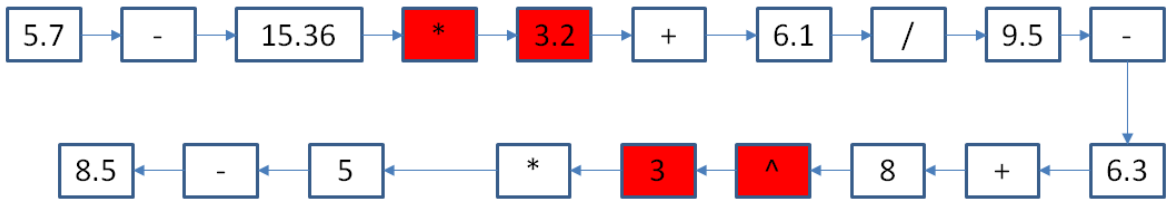


Se encuentra el 2^3, al evaluarse es 8, luego esto afecta a la lista, las piezas que contienen el ^ y el 3 son marcadas como evaluadas (fondo rojo) y **no** serán usadas a futuro. La pieza con el número 2 ahora recibe el valor de 8.

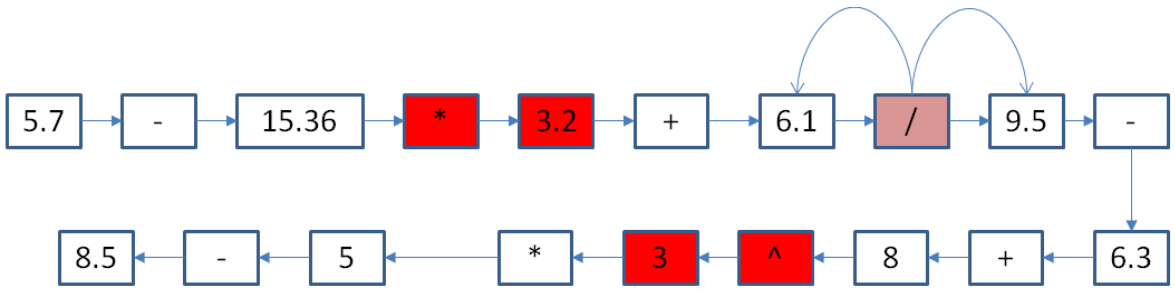
Como no hay más exponentes, el siguiente paso es evaluar la multiplicación y la división:



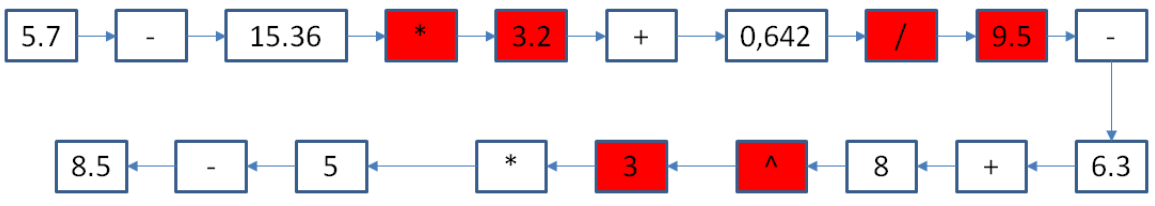
Quedando así la lista (recordar que las piezas marcadas en rojo no se usan luego).



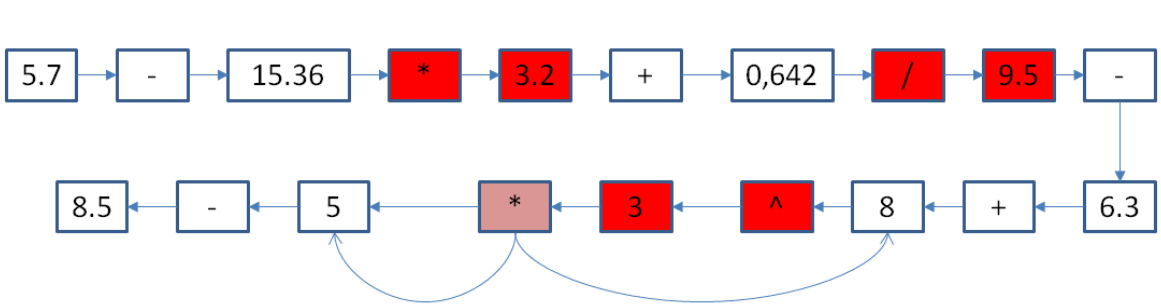
Se sigue buscando multiplicaciones y divisiones



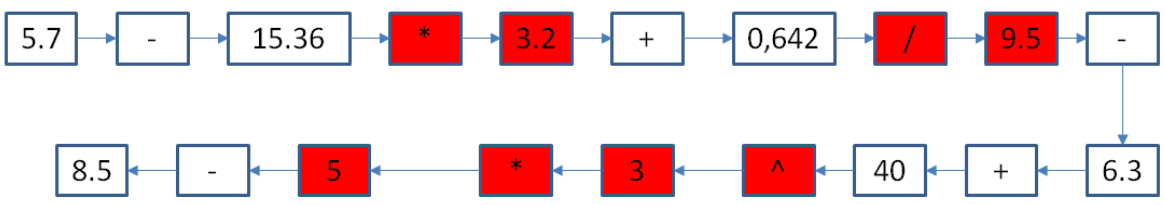
Y evaluando



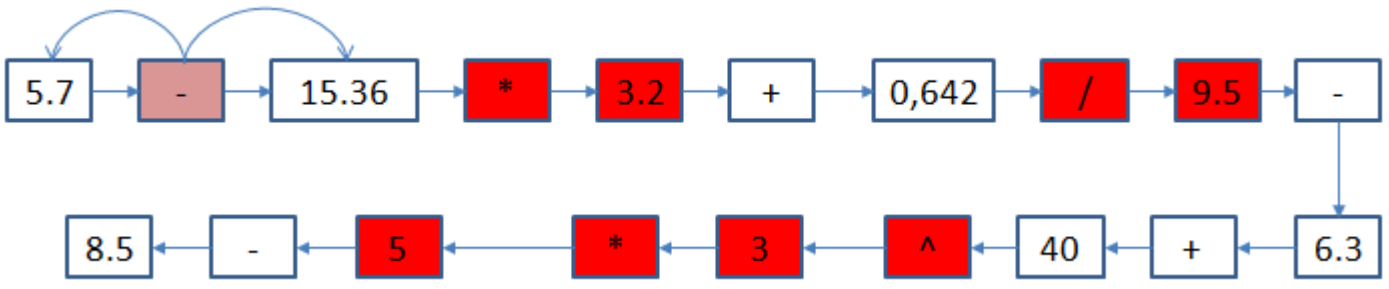
Veamos el caso del último operador multiplicación (*), la pieza anterior está marcada como ya usada, luego tiene que seguir retrocediendo hasta obtener una pieza no usada.

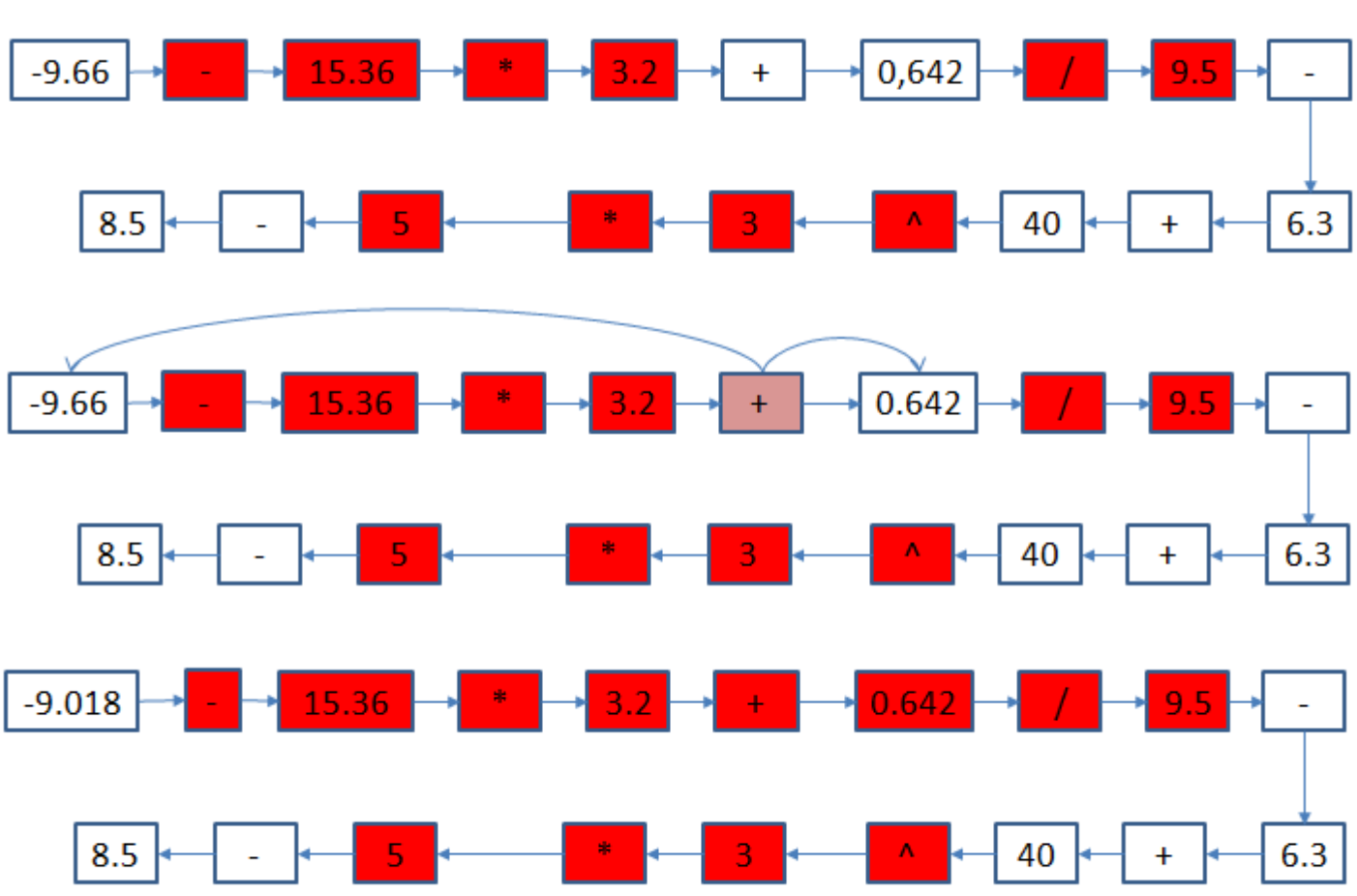


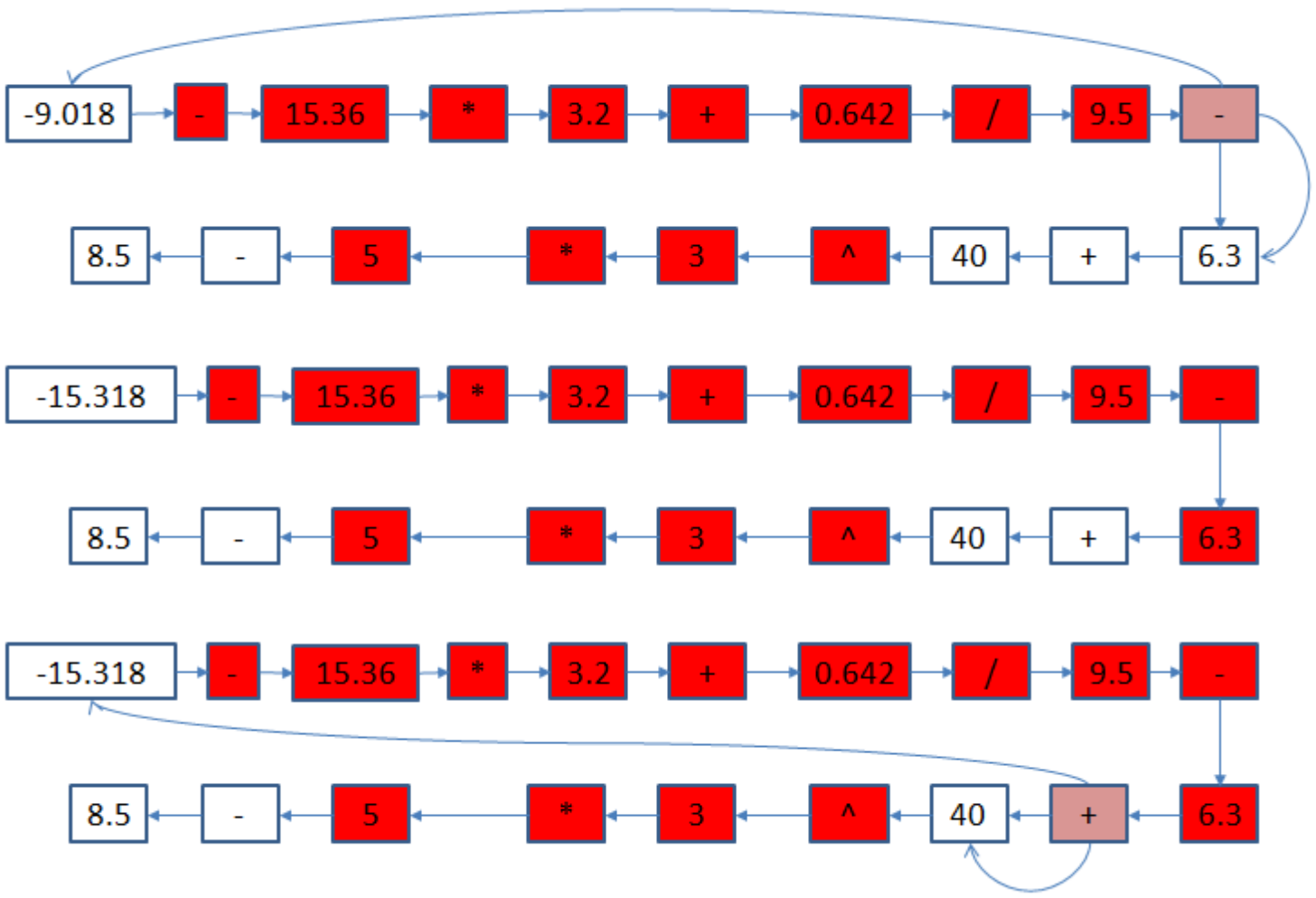
Y evaluar

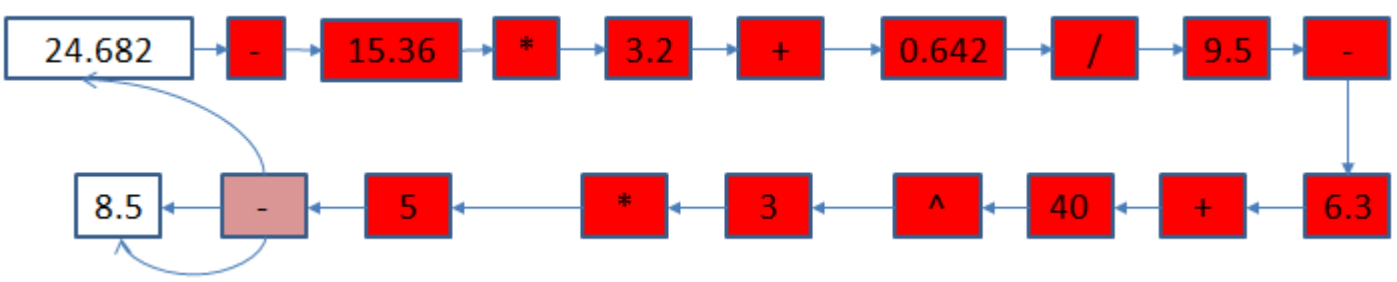


Queda solo operadores suma y resta que se pueden evaluar directamente.

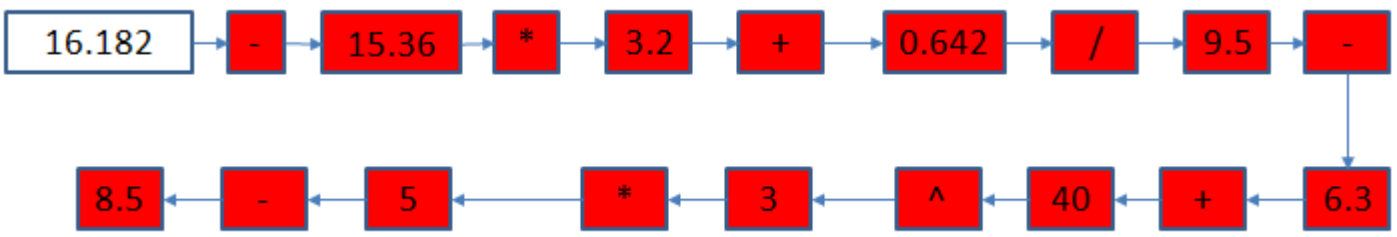




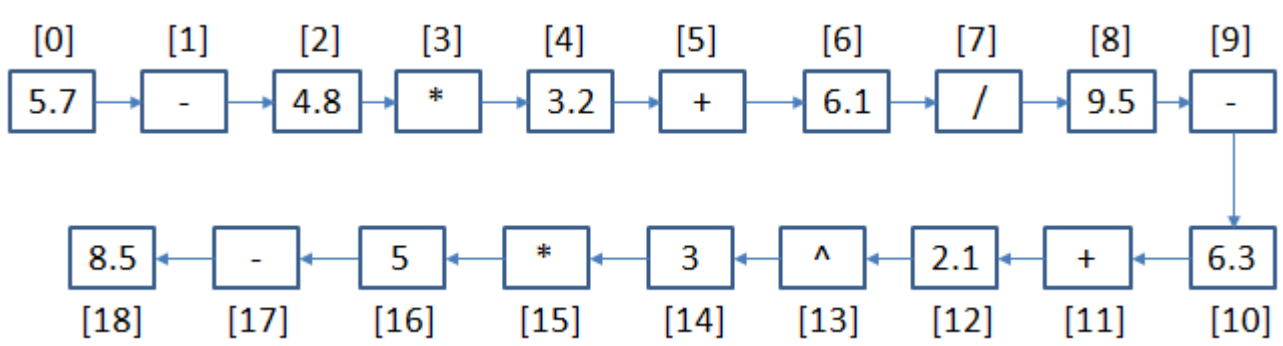




Termina finalmente la evaluación de la expresión simple, todas las piezas a excepción de la primera ya están marcadas en rojo (no se usan) y la primera pieza tiene el resultado de la expresión.



De vuelta a los lenguajes de programación, para hacer esa lista de piezas, usamos una estructura de datos. La que usaremos es la arraylist porque nos permite ir a determinada pieza con saber su ubicación como se ve en la figura:



Las piezas

Cada pieza tiene cinco(5) atributos que son los siguientes:

- **Tipo:** Nos indica si la pieza almacena un número o un operador, teóricamente debería ser un dato de tipo booleano (true es número, false es operador), pero como el evaluador crecerá para manejo de variables y funciones, entonces se usa el tipo de datos entero.
- **Numero:** Almacena el número. De tipo de doble precisión.
- **Operador:** Almacena el operador (+, -, *, /, ^) . De tipo carácter.
- **Evaluado:** Si la pieza está marcada como evaluada tomará el valor true (verdadero). De tipo booleno.
- **Acumula:** Cuando se evalúa la expresión, como se vio en las figuras anteriores, el resultado de la operación se guarda en una pieza (reemplazando el valor que tenía). En la implementación, se guarda el nuevo valor en esta variable de tipo doble precisión porque de esa forma no se pierde la lista original y permite evaluar varias veces la expresión sin necesidad de volverla a construir. ¿Por qué evaluar varias veces? En el caso que la expresión tenga variables que cambian su valor en cada evaluación, un ejemplo, es si se desea hacer una gráfica de la ecuación.

La clase pieza tiene dos constructores:

Primer constructor: Llamado en caso que quiera almacenarse un número. Adicionalmente, copia ese número al atributo Acumula.

Segundo constructor: Llamado en caso que quiera almacenarse un operador

C++

Pieza.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

/* Pieza será la unidad básica de la expresión algebraica, cada nodo tendrá o un operador
   o un número. Este nodo trabaja en una lista simplemente enlazada */
class Pieza
{
private:
      /* 1. Almacena un número, 2. Almacena un operador */
      int tipo;

      /* El número en la expresión algebraica */
      double numero;

      /* El operador (+, -, *, /, ^) en la expresión algebraica */
      char operador;

      /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
            cálculos que determinan el valor de la expresión */
      bool evaluado;

      /* El resultado parcial de la evaluación de la expresión */
      double acumula;

public:
      /* Constructor en el caso que la pieza contenga un número */
      Pieza(double);

      /* Constructor en el caso que la pieza contenga un operador */
      Pieza(char);

      double getAcumula();
      bool isEvaluado();
      double getNumero();
      char getOperador();
      int getTipo();

      void setAcumula(double);
      void setEvaluado(bool);
};
```

Pieza.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <stdio.h>
#include "Pieza.h"

/* Constructor */
Pieza::Pieza(double numero)
{
      this->tipo = 1; //Si el nodo contiene un número real
      this->numero = numero;

      /* Por defecto este nodo no ha sido evaluado */
      this->evaluado = false;

      /* Esta es una copia del valor que será usado cuando se evalúe la expresión */
      this->acumula = numero;
}

/* Constructor */
Pieza::Pieza(char operador)
{
      this->tipo = 2; //Si el nodo contiene un operador (+, -, *, /)
      this->operador = operador;

      /* Por defecto este nodo no ha sido evaluado */
      this->evaluado = false;

      /* No se saca copia de un nodo que contiene un operador */
      this->acumula = 0;
}

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
```

```
double Pieza::getAcumula()
{
      return acumula;
}

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
bool Pieza::isEvaluado()
{
      return evaluado;
}

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
double Pieza::getNumero()
{
      return numero;
}

  /**
   * Retorna el operador (+, -, *, /, ^) que tiene la pieza
   * @return El operador en char
   */
char Pieza::getOperador()
{
      return operador;
}

  /**
   * Retorna que tipo de pieza es: número u operador
   * @return Tipo de pieza
   */
int Pieza::getTipo()
{
      return tipo;
}

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
void Pieza::setAcumula(double acumulado)
{
      this->acumula = acumulado;
}

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
void Pieza::setEvaluado(bool evaluado)
{
      this->evaluado = evaluado;
}
```

EvaluadorExpresionAlgebraica.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include "Pieza.h"
#include <vector>

class EvaluadorExpresionAlgebraica
{
private:
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    static const int ESNUMERO = 1;
    static const int ESOPERADOR = 2;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    static const int ASCIINUMERO = 48;

    /* Lista simplemente enlazada que tiene los componentes (numero u operador
       de la expresión algebraica ya dividida */
    std::vector<Pieza> Piezas;

    /* Variable que se pone a true cuando hay error matemático */
    bool ERRORMATEMATICO;
```

```
public:
    /* Analiza la expresión algebraica y la vuelve una lista */
    void Analizar(char *);

    /* Evalúa la expresión que ha sido previamente analizada */
    double Evaluar(void);

    /* Retorna si hubo error matemático */
    bool getErrorMatematico();
};
```

EvaluadorExpresionAlgebraica.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <math.h>
#include <stdio.h>

#include "EvaluadorExpresionAlgebraica.h"

/**
 * Convierte una expresión simple en una sucesión de nodos.
 * |2| |+| |7| |/| |5| |*| |8| |-| |1.5|
*/
void EvaluadorExpresionAlgebraica::Analizar(char *expr)
{
  /* Inicializa la lista de piezas. */
  Piezas.clear();
  //Piezas.swap(std::vector<Pieza>(Piezas));

  /* Conversión de string a double */
  int entero = 0;
  int fraccion = 0;
  int divide = 1;
  bool puntoentero = false;

  /* Tamaño de la expresión simple */
  int longitud = strlen(expr);

  /* Va extrayendo letra a letra de la expresión simple */
  char letra;

  /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
  bool acumulabaNumero = false;

  for (int cont=0; cont<longitud; cont++)
  {
    /* Trae la letra */
    letra = expr[cont];

    /* Si es un número */
    if( (letra >= '0' && letra <= '9') || letra == '.')
    {
      /* Ir armando el número de tipo double */
      if (letra == '.')
        puntoentero = true;
      else
        if(!puntoentero)  /* puntoentero == false */
          entero = entero * 10 + letra-ASCIINUMERO;
        else
        {
          fraccion = fraccion * 10 + letra-ASCIINUMERO;
          divide *=10;
        }
      acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
         nodo del número si venía acumulando un número */
      if (acumulabaNumero)
      {
        /* El nodo del número */
        Pieza objeto(entero + (double) fraccion/divide);

        /* Agrega a la lista */
        Piezas.push_back(objeto);
      }

      /* El nodo operador */
      Pieza objeto(letra);

      /* Agrega a la lista */
      Piezas.push_back(objeto);
```

```
      /* Inicializa de nuevo las variables de conversión de string a número */
      entero = 0;
      fraccion = 0;
      divide = 1;
      puntoentero = false;
      acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    Pieza objeto(entero + (double) fraccion/divide);
    Piezas.push_back(objeto);
  }

}

/* Ya construida la lista del tipo:
   [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   es ir del operador con mas precedencia al de menos precedencia */
double EvaluadorExpresionAlgebraica::Evaluar()
{
  int pos=0, antes=0, sigue=0;
  ERRORMATEMATICO = false;

  /* Total de nodos en la lista creada */
  int totalPiezas = Piezas.size();

  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Activa todas las piezas para que sean evaluadas */
    Piezas[pos].setEvaluado(false);

    if (Piezas[pos].getTipo() == ESNUMERO)
      Piezas[pos].setAcumula(Piezas[pos].getNumero());
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es exponente */
    if (Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
    {
      /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
      for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
      for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas[pos].setEvaluado(true);
      Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número elevado a Número */
      Piezas[antes].setAcumula(pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es multiplicación o división */
    if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' || Piezas[pos].getOperador() == '/'))
    {
      /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
      for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
      for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas[pos].setEvaluado(true);
      Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número * Número, o Número / Número */
      if (Piezas[pos].getOperador() == '*')
        Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
      else
      {
        if (Piezas[sigue].getAcumula() != 0)
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
        else
        {
          ERRORMATEMATICO = true;
          return 0;
        }
      }
    }
  }
```

```
  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es suma o resta */
    if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' || Piezas[pos].getOperador() == '-'))
    {
      /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
      for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
      for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas[pos].setEvaluado(true);
      Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número + Número, o Número - Número */
      if (Piezas[pos].getOperador() == '+')
        Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
      else
        Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
    }
  }

  /* Resultado de la expresión */
  return Piezas[0].getAcumula();
}


/* Retorna true si hubo un error matemático */
bool EvaluadorExpresionAlgebraica::getErrorMatematico()
{
  return ERRORMATEMATICO;
}
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

Inicio.cpp

```
#include <stdio.h>
#include <string.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
      char expresion[500];
      EvaluadorExpresionAlgebraica objEvaluador;
      double valor, valorreal;

      strcpy(expresion, "8.596*6.871/9.144+2.101-4.597*4^2-1");
      objEvaluador.Analizar(expresion);
      valor = objEvaluador.Evaluar();
      printf("%f\n", valor);
      getchar();
      return 0;
}
```

Visual C# .NET 2010

Pieza.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

using System;
using System.Collections.Generic;

namespace Evaluador1
{
    class Pieza
    {
        /* 1. Almacena un número, 2. Almacena un operador */
        private int tipo;

        /* El número en la expresión algebraica */
        private double numero;

        /* El operador (+, -, *, /, ^) en la expresión algebraica */
        private char operador;

        /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
           cálculos que determinan el valor de la expresión */
        private bool evaluado;

        /* El resultado parcial de la evaluación de la expresión */
        private double acumula;

        /**
         * Constructor en caso que la pieza contenga un número real
         * @param numero Número de tipo double
         */
        public Pieza(double numero)
        {
            this.tipo = 1; //Es un número
            this.numero = numero;

            this.evaluado = false;
            this.acumula = numero;
        }


        /**
         * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
         * @param operador Que puede ser +, -, *, /, ^
         */
        public Pieza(char operador)
        {
            this.tipo = 2; //Es un operador
            this.operador = operador;

            this.evaluado = false;
            this.acumula = 0;
        }

        /** Retorna el acumulado que tiene esta pieza
         * @return Acumulado de la pieza
         */
        public double getAcumula()
        {
            return acumula;
        }

        /** Retorna si la pieza ya fue evaluada
         * @return True si la pieza ya fue evaluada
         */
        public bool isEvaluado()
        {
            return evaluado;
        }

        /**
         * Retorna el número de tipo double que tiene la pieza
         * @return El valor del número tipo double
         */
        public double getNumero()
        {
            return numero;
        }

        /**
         * Retorna el operador (+, -, *, /, ^) que tiene la pieza
         * @return El operador en char
         */
```

```
        public char getOperador()
        {
            return operador;
        }

        /**
         * Retorna que tipo de pieza es: número u operador
         * @return Tipo de pieza
         */
        public int getTipo()
        {
            return tipo;
        }

        /**
         * Da valor al acumulado por pieza
         * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
         */
        public void setAcumula(double acumulado)
        {
            this.acumula = acumulado;
        }

        /**
         * Marca la pieza como ya evaluada
         * @param evaluado true si la pieza ya ha sido evaluada, false si no
         */
        public void setEvaluado(bool evaluado)
        {
            this.evaluado = evaluado;
        }
    }
}
```

EvaluadorExpresionAlgebraica.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

using System;
using System.Collections.Generic;

namespace Evaluador1
{
    class EvaluadorExpresionAlgebraica
    {
      /**
       * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
       */
      private static int ESNUMERO = 1;
      private static int ESOPERADOR = 2;

      /**
       * Esta constante sirve para que se reste al carácter y se obtenga el número
       * Ejemplo:  '7' - ASCIINUMERO =  7
       */
      private static int ASCIINUMERO = 48;

      /**
       * Lista simplemente enlazada que tiene los componentes (número u operador)
       * de la expresión algebraica ya dividida.
       */
      private List<Pieza> Piezas = new List<Pieza>();

      /**
       * Variable que se pone a true cuando hay un error matemático
       */
      private bool ERRORMATEMATICO;

      /**
       * Este método se encarga de analizar la expresión y convertirla en una
       * estructura que permita evaluar la expresión.
       *
       * Convierte una expresión algebraica en una sucesión de nodos.
       * |2| |+| |8.3| |/| |5| |*| |7.4|
       *
       * @param expr La expresión algebraica sin espacios y en minúsculas
      */
      public void Analizar(String expr)
      {
        /* Inicializa la lista de piezas  */
        Piezas.Clear();

        /* Tamaño de la expresión simple */
        int longitud = expr.Length;
```

```
        /* Va extrayendo letra a letra de la expresión simple */
        char letra;

        /* Conversión de string a double */
        int entero = 0, fraccion = 0, divide = 1;
        bool puntoentero = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        bool acumulabaNumero = false;

        for (int cont=0; cont<longitud; cont++)
        {
          /* Trae la letra */
          letra = expr[cont];

          /* Si es un número */
          if( (letra >= '0' && letra <= '9') || letra == '.')
          {
            /* Ir armando el número de tipo double */
            if (letra == '.')
              puntoentero = true;
            else
              if(!puntoentero)  /* puntoentero == false */
                entero = entero * 10 + letra-ASCIINUMERO;
              else
              {
                fraccion = fraccion * 10 + letra-ASCIINUMERO;
                divide *=10;
              }
              acumulabaNumero = true;
          }
          else
          {
            /* Si es un operador entonces crea el nodo del operador y el
              nodo del número si venía acumulando un número */
            if (acumulabaNumero)
              Piezas.Add(new Pieza(entero + (double) fraccion/divide));

            /* Agrega a la lista el operador */
            Piezas.Add(new Pieza(letra));

            /* Inicializa de nuevo las variables de conversión de string a número */
            entero = fraccion = 0;
            divide = 1;
            puntoentero = acumulabaNumero = false;
          }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
        {
          Pieza objeto = new Pieza(entero + (double) fraccion/divide);
          Piezas.Add(objeto);
        }
      }

      /**
       * Ya construida la lista del tipo:
       * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
       * es ir del operador con mas precedencia al de menos precedencia.
       *
       * Este método se llama después de haber sido analizada la expresión.
       *
       * @return El valor de la expresión evaluada (double)
       */
      public double Evaluar()
      {
        int pos=0, antes=0, sigue=0;
        ERRORMATEMATICO = false;

        /* Total de nodos en la lista creada */
        int totalPiezas = Piezas.Count;

        for (pos=0; pos<totalPiezas; pos++)
        {
          /* Activa todas las piezas para que sean evaluadas */
          Piezas[pos].setEvaluado(false);

          if (Piezas[pos].getTipo() == ESNUMERO)
            Piezas[pos].setAcumula(Piezas[pos].getNumero());
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
          /* Si encuentra un nodo del tipo operador y es exponente */
          if (Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
          {
            /* Busca un nodo anterior que sea número y no haya sido evaluado */
            for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

            /* Busca un nodo siguiente que sea número y no haya sido evaluado */
```

```
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número elevado a Número */
        Piezas[antes].setAcumula(Math.Pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es multiplicación o división */
      if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' || Piezas[pos].getOperador() == '/'))
      {
        /* Busca un nodo anterior que sea número y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número * Número, o Número / Número */
        if (Piezas[pos].getOperador() == '*')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
        else
        {
          if (Piezas[sigue].getAcumula() != 0)
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
        }
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
        /* Si encuentra un nodo del tipo operador y es suma o resta */
        if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' || Piezas[pos].getOperador() == '-'))
        {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas[pos].setEvaluado(true);
          Piezas[sigue].setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if (Piezas[pos].getOperador() == '+')
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
          else
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
        }
      }

      /* Resultado de la expresión */
      return Piezas[0].getAcumula();
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    public bool getErrorMatematico()
    {
        return ERRORMATEMATICO;
    }
  }
}
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace Evaluador1
{
    class Program
    {
        static void Main(string[] args)
        {
            EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
            string expresion;
            double valor, valorreal;

            expresion = "1.64+1.440-0.952*5^2+4";
            objEvaluador.Analizar(expresion);
            valor = objEvaluador.Evaluar();
            Console.WriteLine(valor);
            Console.ReadKey();
        }
    }
}
```

Visual Basic .NET 2010

Pieza.vb

```
' Autor: Rafael Alberto Moreno Parra
' Correo: ramsoftware@gmail.com
' Página Web: http://darwin.50webs.com
' Licencia: LGPL
' Fecha: Enero de 2012

Imports System.Collections.Generic

Public Class Pieza
    ' 1. Almacena un número, 2. Almacena un operador
    Private tipo As Integer

    ' El número en la expresión algebraica
    Private numero As Double

    ' El operador (+, -, *, /, ^) en la expresión algebraica
    Private operador As Char

    ' Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    ' cálculos que determinan el valor de la expresión
    Private evaluado As Boolean

    ' El resultado parcial de la evaluación de la expresión
    Private acumula As Double

    ' Constructor en caso que la pieza contenga un número real
    ' @param numero Número de tipo double
    Public Sub New(ByVal numero As Double)
        Me.tipo = 1
        'Es un número
        Me.numero = numero

        Me.evaluado = False
        Me.acumula = numero
    End Sub

    ' Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
    ' @param operador Que puede ser +, -, *, /, ^
    Public Sub New(ByVal operador As Char)
        Me.tipo = 2
        'Es un operador
        Me.operador = operador

        Me.evaluado = False
        Me.acumula = 0
    End Sub

    ' Retorna el acumulado que tiene esta pieza
    ' @return Acumulado de la pieza
    Public Function getAcumula() As Double
        Return acumula
    End Function

    ' Retorna si la pieza ya fue evaluada
    ' @return True si la pieza ya fue evaluada
    Public Function isEvaluado() As Boolean
        Return evaluado
    End Function

    ' Retorna el número de tipo double que tiene la pieza
    ' @return El valor del número tipo double
    Public Function getNumero() As Double
        Return numero
    End Function

    ' Retorna el operador (+, -, *, /, ^) que tiene la pieza
    ' @return El operador en char
    Public Function getOperador() As Char
        Return operador
    End Function

    ' Retorna que tipo de pieza es: número u operador
    ' @return Tipo de pieza
    Public Function getTipo() As Integer
        Return tipo
    End Function

    ' Da valor al acumulado por pieza
    ' @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
    Public Sub setAcumula(ByVal acumulado As Double)
        Me.acumula = acumulado
    End Sub

    ' Marca la pieza como ya evaluada
    ' @param evaluado true si la pieza ya ha sido evaluada, false si no
    Public Sub setEvaluado(ByVal evaluado As Boolean)
        Me.evaluado = evaluado
    End Sub
End Class
```

EvaluadorExpresionAlgebraica.vb

```
' Autor: Rafael Alberto Moreno Parra
' Correo: ramsoftware@gmail.com
' Página Web: http://darwin.50webs.com
' Licencia: LGPL
' Fecha: Enero de 2012

Imports System.Collections.Generic

Class EvaluadorExpresionAlgebraica

  ' Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
  Private Shared ESNUMERO As Integer = 1
  Private Shared ESOPERADOR As Integer = 2

  ' Esta constante sirve para que se reste al carácter y se obtenga el número
  ' Ejemplo:  '7' - ASCIINUMERO =  7
  Private Shared ASCIINUMERO As Integer = 48

  ' Lista simplemente enlazada que tiene los componentes (número u operador)
  ' de la expresión algebraica ya dividida.
  Private Piezas As New List(Of Pieza)()

  ' Variable que se pone a true cuando hay un error matemático
  Private ERRORMATEMATICO As Boolean

' Este método se encarga de analizar la expresión y convertirla en una
' estructura que permita evaluar la expresión.
'
' Convierte una expresión algebraica en una sucesión de nodos.
' |2| |+| |8.3| |/| |5| |*| |7.4|
'
' @param expr La expresión algebraica sin espacios y en minúsculas
Public Sub Analizar(ByVal expr As [String])
  ' Inicializa la lista de piezas
  Piezas.Clear()

  ' Tamaño de la expresión simple
  Dim longitud As Integer = expr.Length

  ' Va extrayendo letra a letra de la expresión simple
  Dim letra As Char

  ' Conversión de string a double
  Dim entero As Integer = 0, fraccion As Integer = 0, divide As Integer = 1
  Dim puntoentero As Boolean = False

  ' Si llevaba acumulando un valor numérico esta variable se vuelve true
  Dim acumulabaNumero As Boolean = False

  For cont As Integer = 0 To longitud - 1
    ' Trae la letra
    letra = expr(cont)

    ' Si es un número
    If (letra >= "0" AndAlso letra <= "9") OrElse letra = "." Then
      ' Ir armando el número de tipo double

      If letra = "." Then
        puntoentero = True
      ElseIf Not puntoentero Then
        ' puntoentero == false
        entero = entero * 10 + Asc(letra) - ASCIINUMERO
      Else
        fraccion = fraccion * 10 + Asc(letra) - ASCIINUMERO
        divide *= 10
      End If
      acumulabaNumero = True
    Else
      ' Si es un operador entonces crea el nodo del operador y el
      '     nodo del número si venía acumulando un número
      If acumulabaNumero Then
        Piezas.Add(New Pieza(entero + CDbl(fraccion) / divide))
      End If

      ' Agrega a la lista el operador
      Piezas.Add(New Pieza(letra))

      ' Inicializa de nuevo las variables de conversión de string a número
      entero = 0
      fraccion = 0
      divide = 1
      puntoentero = False
      acumulabaNumero = False
    End If
  Next

  ' Cierra la expresión simple preguntando si el último operando es un número
  If acumulabaNumero Then
```

```
      Dim objeto As New Pieza(entero + CDbl(fraccion) / divide)
      Piezas.Add(objeto)
    End If
  End Sub


  ' Ya construida la lista del tipo:
  ' [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
  ' es ir del operador con mas precedencia al de menos precedencia.
  '
  ' Este método se llama después de haber sido analizada la expresión.
  '
  ' @return El valor de la expresión evaluada (double)
  Public Function Evaluar() As Double
    Dim pos As Integer = 0, antes As Integer = 0, sigue As Integer = 0
    ERRORMATEMATICO = False

    ' Total de nodos en la lista creada

    Dim totalPiezas As Integer = Piezas.Count

    For pos = 0 To totalPiezas - 1
      ' Activa todas las piezas para que sean evaluadas

      Piezas(pos).setEvaluado(False)

      If Piezas(pos).getTipo() = ESNUMERO Then
        Piezas(pos).setAcumula(Piezas(pos).getNumero())
      End If
    Next

    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
      ' Si encuentra un nodo del tipo operador y es exponente
      If Piezas(pos).getTipo() = ESOPERADOR AndAlso Piezas(pos).getOperador() = "^" Then
        ' Busca un nodo anterior que sea número y no haya sido evaluado
        antes = pos - 1
        While Piezas(antes).isEvaluado()
          antes -= 1
        End While

        ' Busca un nodo siguiente que sea número y no haya sido evaluado
        sigue = pos + 1
        While Piezas(sigue).isEvaluado()
          sigue += 1
        End While

        ' Marca esos nodos actual y siguiente como ya evaluados
        Piezas(pos).setEvaluado(True)
        Piezas(sigue).setEvaluado(True)

        ' Hace la operación de Número elevado a Número
        Piezas(antes).setAcumula(Math.Pow(Piezas(antes).getAcumula(), Piezas(sigue).getAcumula()))
      End If
    Next

    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
      ' Si encuentra un nodo del tipo operador y es multiplicación o división
      If Piezas(pos).getTipo() = ESOPERADOR AndAlso (Piezas(pos).getOperador() = "*" OrElse Piezas(pos).getOperador() = "/")
Then
        ' Busca un nodo anterior que sea número y no haya sido evaluado
        antes = pos - 1
        While Piezas(antes).isEvaluado()
          antes -= 1
        End While

        ' Busca un nodo siguiente que sea número y no haya sido evaluado
        sigue = pos + 1
        While Piezas(sigue).isEvaluado()
          sigue += 1
        End While

        ' Marca esos nodos actual y siguiente como ya evaluados
        Piezas(pos).setEvaluado(True)
        Piezas(sigue).setEvaluado(True)

        ' Hace la operación de Número * Número, o Número / Número
        If Piezas(pos).getOperador() = "*" Then
          Piezas(antes).setAcumula(Piezas(antes).getAcumula() * Piezas(sigue).getAcumula())
        Else
          If Piezas(sigue).getAcumula() <> 0 Then
            Piezas(antes).setAcumula(Piezas(antes).getAcumula() / Piezas(sigue).getAcumula())
          Else
            ERRORMATEMATICO = True
            Return 0
          End If
        End If
      End If
    Next

    ' Recorre toda la lista
```

```
    For pos = 0 To totalPiezas - 1
      ' Si encuentra un nodo del tipo operador y es suma o resta
      If Piezas(pos).getTipo() = ESOPERADOR AndAlso (Piezas(pos).getOperador() = "+" OrElse Piezas(pos).getOperador() = "-")
Then
        ' Busca un nodo anterior que sea número y no haya sido evaluado
        antes = pos - 1
        While Piezas(antes).isEvaluado()
          antes -= 1
        End While

        ' Busca un nodo siguiente que sea número y no haya sido evaluado
        sigue = pos + 1
        While Piezas(sigue).isEvaluado()
          sigue += 1
        End While

        ' Marca esos nodos actual y siguiente como ya evaluados
        Piezas(pos).setEvaluado(True)
        Piezas(sigue).setEvaluado(True)

        ' Hace la operación de Número + Número, o Número - Número
        If Piezas(pos).getOperador() = "+" Then
          Piezas(antes).setAcumula(Piezas(antes).getAcumula() + Piezas(sigue).getAcumula())
        Else
          Piezas(antes).setAcumula(Piezas(antes).getAcumula() - Piezas(sigue).getAcumula())
        End If
      End If
    Next

    ' Resultado de la expresión

    Return Piezas(0).getAcumula()
  End Function


  ' Retorna true si hubo un error matemático
  ' @return TRUE si hubo un error matemático
  Public Function getErrorMatematico() As Boolean
    Return ERRORMATEMATICO
  End Function
End Class
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

Module1.vb

```
Module Module1

    Sub Main()
        Dim objEvaluador As New EvaluadorExpresionAlgebraica
        Dim expresion As String
        Dim valor, valorreal As Double

        expresion = "7.587+0.460*1.432*5^2-1"
        objEvaluador.Analizar(expresion)
        valor = objEvaluador.Evaluar()
        Console.WriteLine(valor)

        Console.ReadKey()

    End Sub

End Module
```

Java

Pieza.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
public class Pieza
{
  /* 1. Almacena un número, 2. Almacena un operador */
  private int tipo;

  /* El número en la expresión algebraica */
  private double numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  private char operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  private boolean evaluado;

  /* El resultado parcial de la evaluación de la expresión */
  private double acumula;

  /**
   * Constructor en caso que la pieza contenga un número real
   * @param numero Número de tipo double
   */
  public Pieza(double numero)
  {
   this.tipo = 1; //Es un número
   this.numero = numero;

   this.evaluado = false;
   this.acumula = numero;
  }

  /**
   * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
   * @param operador Que puede ser +, -, *, /, ^
   */
  public Pieza(char operador)
  {
    this.tipo = 2; //Es un operador
    this.operador = operador;

    this.evaluado = false;
    this.acumula = 0;
  }

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
  public double getAcumula()
  {
    return acumula;
  }

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
  public boolean isEvaluado()
  {
    return evaluado;
  }

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
  public double getNumero()
  {
    return numero;
  }

  /**
   * Retorna el operador (+, -, *, /, ^) que tiene la pieza
   * @return El operador en char
   */
  public char getOperador()
  {
    return operador;
  }

  /**
   * Retorna que tipo de pieza es: número u operador
   * @return Tipo de pieza
   */
```

```
  public int getTipo()
  {
    return tipo;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  public void setAcumula(double acumulado)
  {
    this.acumula = acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  void setEvaluado(boolean evaluado)
  {
    this.evaluado = evaluado;
  }
}
```

EvaluadorExpresionAlgebraica.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
import java.util.ArrayList;

public class EvaluadorExpresionAlgebraica
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  private static final int ESNUMERO = 1;
  private static final int ESOPERADOR = 2;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * Ejemplo:  '7' - ASCIINUMERO =  7
   */
  private static final int ASCIINUMERO = 48;

  /**
   * Lista simplemente enlazada que tiene los componentes (número u operador)
   * de la expresión algebraica ya dividida.
   */
  private ArrayList<Pieza> Piezas = new ArrayList<Pieza>();

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  private boolean ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |8.3| |/| |5| |*| |7.4|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  public void Analizar(String expr)
  {
    /* Inicializa la lista de piezas  */
    Piezas.clear();

    /* Tamaño de la expresión simple */
    int longitud = expr.length();

    /* Va extrayendo letra a letra de la expresión simple */
    char letra;

    /* Conversión de string a double */
    int entero = 0, fraccion = 0, divide = 1;
    boolean puntoentero = false;

    /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
    boolean acumulabaNumero = false;

    for (int cont=0; cont<longitud; cont++)
    {
      /* Trae la letra */
      letra = expr.charAt(cont);
```

```
    /* Si es un número */
    if( (letra >= '0' && letra <= '9') || letra == '.')
    {
      /* Ir armando el número de tipo double */
      if (letra == '.')
        puntoentero = true;
      else
        if(!puntoentero)  /* puntoentero == false */
          entero = entero * 10 + letra-ASCIINUMERO;
        else
        {
          fraccion = fraccion * 10 + letra-ASCIINUMERO;
          divide *=10;
        }
        acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
        nodo del número si venía acumulando un número */
      if (acumulabaNumero)
        Piezas.add(new Pieza(entero + (double) fraccion/divide));

      /* Agrega a la lista */
      Piezas.add(new Pieza(letra));

      /* Inicializa de nuevo las variables de conversión de string a número */
      entero = fraccion = 0;
      divide = 1;
      puntoentero = acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    Pieza objeto = new Pieza(entero + (double) fraccion/divide);
    Piezas.add(objeto);
  }
}

/**
 * Ya construida la lista del tipo:
 * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
public double Evaluar()
{
  int pos=0, antes=0, sigue=0;
  ERRORMATEMATICO = false;

  /* Total de nodos en la lista creada */
  int totalPiezas = Piezas.size();

  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Activa todas las piezas para que sean evaluadas */
    Piezas.get(pos).setEvaluado(false);

    if (Piezas.get(pos).getTipo() == ESNUMERO)
      Piezas.get(pos).setAcumula(Piezas.get(pos).getNumero());
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es exponente */
    if (Piezas.get(pos).getTipo() == ESOPERADOR && Piezas.get(pos).getOperador() == '^')
    {
      /* Busca un nodo anterior que sea número y no haya sido evaluado */
      for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número y no haya sido evaluado */
      for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas.get(pos).setEvaluado(true);
      Piezas.get(sigue).setEvaluado(true);

      /* Hace la operación de Número elevado a Número */
      Piezas.get(antes).setAcumula(Math.pow(Piezas.get(antes).getAcumula(), Piezas.get(sigue).getAcumula()));
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
```

```
      /* Si encuentra un nodo del tipo operador y es multiplicación o división */
      if (Piezas.get(pos).getTipo() == ESOPERADOR && (Piezas.get(pos).getOperador() == '*' || Piezas.get(pos).getOperador() ==
'/'))
      {
        /* Busca un nodo anterior que sea número y no haya sido evaluado */
        for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número y no haya sido evaluado */
        for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas.get(pos).setEvaluado(true);
        Piezas.get(sigue).setEvaluado(true);

        /* Hace la operación de Número * Número, o Número / Número */
        if (Piezas.get(pos).getOperador() == '*')
          Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() *  Piezas.get(sigue).getAcumula() );
        else
        {
          if (Piezas.get(sigue).getAcumula() != 0)
            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() /  Piezas.get(sigue).getAcumula() );
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
        }
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
        /* Si encuentra un nodo del tipo operador y es suma o resta */
        if (Piezas.get(pos).getTipo() == ESOPERADOR && (Piezas.get(pos).getOperador() == '+' || Piezas.get(pos).getOperador() ==
'-'))
        {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas.get(pos).setEvaluado(true);
          Piezas.get(sigue).setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if (Piezas.get(pos).getOperador() == '+')
            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() +  Piezas.get(sigue).getAcumula() );
          else
            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() -  Piezas.get(sigue).getAcumula() );
        }
      }

    /* Resultado de la expresión */
    return Piezas.get(0).getAcumula();
   }

  /**
   * Retorna true si hubo un error matemático
   * @return TRUE si hubo un error matemático
   */
  public boolean getErrorMatematico()
  {
      return ERRORMATEMATICO;
  }
}
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

Inicio.java

```
public class Inicio
{
    public static void main(String args[])
    {
      EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
      String expresion;
      double valor, valorreal;
      expresion = "3.398+1.609/5.478-3.529+7^3*4.248";
      objEvaluador.Analizar(expresion);
      valor = objEvaluador.Evaluar();
      System.out.println(valor);
    }
}
```

PHP

Pieza.php

```php
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

class Pieza
{
  /* 1. Almacena un número, 2. Almacena un operador */
  var $tipo;

  /* El número en la expresión algebraica */
  var $numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  var $operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  var $evaluado;

  /* El resultado parcial de la evaluación de la expresión */
  var $acumula;

  /**
   * Constructor en caso que la pieza contenga un número real
   * @param numero Número de tipo double
   */
  public function ConstructorNumero($numero)
  {
   $this->tipo = 1; //Es un número
   $this->numero = $numero;

   $this->evaluado = false;
   $this->acumula = $numero;
  }

  /**
   * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
   * @param operador Que puede ser +, -, *, /, ^
   */
  public function ConstructorOperador($operador)
  {
    $this->tipo = 2; //Es un operador
    $this->operador = $operador;

    $this->evaluado = false;
    $this->acumula = 0;
  }

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
  public function getAcumula()
  {
    return $this->acumula;
  }

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
  function isEvaluado()
  {
    return $this->evaluado;
  }

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
  public function getNumero()
  {
    return $this->numero;
  }

  /**
   * Retorna el operador (+, -, *, /, ^) que tiene la pieza
   * @return El operador en char
   */
  public function getOperador()
  {
    return $this->operador;
  }

  /**
   * Retorna que tipo de pieza es: número u operador
```

```
   * @return Tipo de pieza
   */
  public function getTipo()
  {
    return $this->tipo;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  public function setAcumula($acumulado)
  {
    $this->acumula = $acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  public function setEvaluado($evaluado)
  {
    $this->evaluado = $evaluado;
  }
}
?>
```

EvaluadorExpresionAlgebraica.php

```
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
include("Pieza.php");

class EvaluadorExpresionAlgebraica
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  var $ESNUMERO = 1;
  var $ESOPERADOR = 2;

  /**
   * Lista simplemente enlazada que tiene los componentes (número u operador)
   * de la expresión algebraica ya dividida.
   */
  var $Piezas = array();

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  var $ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |8.3| |/| |5| |*| |7.4|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  public function Analizar($expr)
  {
    /* Inicializa la lista de piezas  */
    unset($this->Piezas);

    /* Tamaño de la expresión simple */
    $longitud = strlen($expr);

    /* Conversión de string a double */
    $entero = 0;
      $fraccion = 0;
      $divide = 1;
    $puntoentero = false;

    /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
    $acumulabaNumero = false;

    for ($cont=0; $cont<$longitud; $cont++)
    {
      /* Trae la letra */
      $letra = $expr{$cont};

      /* Si es un número */
      if( ($letra >= '0' && $letra <= '9') || $letra == '.')
```

```
    {
      /* Ir armando el número de tipo double */
      if ($letra == '.')
        $puntoentero = true;
      else
        if(!$puntoentero)  /* $puntoentero == false */
          $entero = $entero * 10 + $letra;
        else
        {
          $fraccion = $fraccion * 10 + $letra;
          $divide *=10;
        }
        $acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
        nodo del número si venía acumulando un número */
      if ($acumulabaNumero)
          {
            $objeto = new Pieza();
            $objeto->ConstructorNumero($entero + $fraccion/$divide);
        $this->Piezas[] = $objeto;
          }

      /* Agrega a la lista */
          $objeto = new Pieza();
          $objeto->ConstructorOperador($letra);
      $this->Piezas[] = $objeto;

          /* Inicializa de nuevo las variables de conversión de string a número */
          $entero = 0;
              $fraccion = 0;
          $divide = 1;
          $puntoentero = false;
              $acumulabaNumero = false;
        }
      }

      /* Cierra la expresión simple preguntando si el último operando es un número */
      if ($acumulabaNumero)
      {
        $objeto = new Pieza();
          $objeto->ConstructorNumero($entero + $fraccion/$divide);
        $this->Piezas[] = $objeto;
      }


    }

    /**
     * Ya construida la lista del tipo:
     * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
     * es ir del operador con mas precedencia al de menos precedencia.
     *
     * Este método se llama después de haber sido analizada la expresión.
     *
     * @return El valor de la expresión evaluada (double)
     */
    public function Evaluar()
    {
      $pos=0;
      $antes=0;
      $sigue=0;
      $this->ERRORMATEMATICO = false;

      /* Total de nodos en la lista creada */
      $totalPiezas = sizeof($this->Piezas);

      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
        /* Activa todas las piezas para que sean evaluadas */
        $this->Piezas[$pos]->setEvaluado(false);

        if ($this->Piezas[$pos]->getTipo() == $this->ESNUMERO)
          $this->Piezas[$pos]->setAcumula($this->Piezas[$pos]->getNumero());
      }

      /* Recorre toda la lista */
      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
        /* Si encuentra un nodo del tipo operador y es exponente */
        if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && $this->Piezas[$pos]->getOperador() == '^')
        {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          $this->Piezas[$pos]->setEvaluado(true);
```

```
        $this->Piezas[$sigue]->setEvaluado(true);

        /* Hace la operación de Número elevado a Número */
        $this->Piezas[$antes]->setAcumula(pow($this->Piezas[$antes]->getAcumula(), $this->Piezas[$sigue]->getAcumula()));
      }
    }

    /* Recorre toda la lista */
    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
      /* Si encuentra un nodo del tipo operador y es multiplicación o división */
      if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this->Piezas[$pos]->getOperador() == '*' || $this-
>Piezas[$pos]->getOperador() == '/'))
      {
        /* Busca un nodo anterior que sea número y no haya sido evaluado */
        for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

        /* Busca un nodo siguiente que sea número y no haya sido evaluado */
        for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        $this->Piezas[$pos]->setEvaluado(true);
        $this->Piezas[$sigue]->setEvaluado(true);

        /* Hace la operación de Número * Número, o Número / Número */
        if ($this->Piezas[$pos]->getOperador() == '*')
          $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() *  $this->Piezas[$sigue]->getAcumula() );
        else
        {
          if ($this->Piezas[$sigue]->getAcumula() != 0)
            $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() /  $this->Piezas[$sigue]->getAcumula() );
          else
          {
            $this->ERRORMATEMATICO = true;
            return 0;
          }
        }
      }
    }

    /* Recorre toda la lista */
    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
        /* Si encuentra un nodo del tipo operador y es suma o resta */
        if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this->Piezas[$pos]->getOperador() == '+' || $this-
>Piezas[$pos]->getOperador() == '-'))
        {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          $this->Piezas[$pos]->setEvaluado(true);
          $this->Piezas[$sigue]->setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if ($this->Piezas[$pos]->getOperador() == '+')
            $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() +  $this->Piezas[$sigue]->getAcumula() );
              else
            $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() -  $this->Piezas[$sigue]->getAcumula() );
        }
      }

    /* Resultado de la expresión */
    return $this->Piezas[0]->getAcumula();
  }

  /**
   * Retorna true si hubo un error matemático
   * @return TRUE si hubo un error matemático
   */
  public function getERRORMATEMATICO()
  {
      return $this->ERRORMATEMATICO;
  }
}
?>
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

index.php

```php
<?php
include("EvaluadorExpresionAlgebraica.php");

$objEvaluador = new EvaluadorExpresionAlgebraica();

$expresion = '1.624-4.143+7^2+6';
$objEvaluador->Analizar($expresion);
$valor = $objEvaluador->Evaluar();
echo $valor;
?>
```

JAVASCRIPT

EvaluadorExpresionAlgebraica.js

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
function Pieza()
{
  /* 1. Almacena un número, 2. Almacena un operador */
  this.tipo;

  /* El número en la expresión algebraica */
  this.numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  this.operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  this.evaluado;

  /* El resultado parcial de la evaluación de la expresión */
  this.acumula;

  /**
   * Constructor en caso que la pieza contenga un número real
   * @param numero Número de tipo double
   */
  this.ConstructorNumero = function(numero)
  {
    this.tipo = 1; //Es un número
    this.numero = numero;

    this.evaluado = false;
    this.acumula = numero;
  }

  /**
   * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
   * @param operador Que puede ser +, -, *, /, ^
   */
  this.ConstructorOperador = function(operador)
  {
    this.tipo = 2; //Es un operador
    this.operador = operador;

    this.evaluado = false;
    this.acumula = 0;
  }

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
  this.getAcumula = function()
  {
    return this.acumula;
  }

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
  this.isEvaluado = function()
  {
    return this.evaluado;
  }

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
  this.getNumero = function()
  {
    return this.numero;
  }

  /**
   * Retorna el operador (+, -, *, /, ^) que tiene la pieza
   * @return El operador en char
   */
  this.getOperador = function()
  {
    return this.operador;
  }

  /**
   * Retorna que tipo de pieza es: número u operador
   * @return Tipo de pieza
   */
```

```
  this.getTipo = function()
  {
    return this.tipo;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  this.setAcumula = function(acumulado)
  {
    this.acumula = acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  this.setEvaluado = function(evaluado)
  {
    this.evaluado = evaluado;
  }
}

function EvaluadorExpresionAlgebraica()
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  this.ESNUMERO = 1;
  this.ESOPERADOR = 2;

  /**
   * Lista simplemente enlazada que tiene los componentes (número u operador)
   * de la expresión algebraica ya dividida.
   */
  this.Piezas = [];

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  this.ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |8.3| |/| |5| |*| |7.4|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  this.Analizar = function(expr)
  {
    /* Inicializa la lista de piezas  */
    this.Piezas.length = 0;
    NumeroPieza = 0;

    /* Tamaño de la expresión simple */
    longitud = expr.length;

    /* Conversión de string a double */
    entero = 0;
    fraccion = 0;
    divide = 1;
    puntoentero = false;

    /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
    acumulabaNumero = false;

    for (cont=0; cont<longitud; cont++)
    {
      /* Trae la letra */
      letra = expr.charAt(cont);

      /* Si es un número */
      if( (letra >= '0' && letra <= '9') || letra == '.')
      {
        /* Ir armando el número de tipo double */
        if (letra == '.')
          puntoentero = true;
        else
          if(!puntoentero)  /* puntoentero == false */
            entero = entero * 10 + parseFloat(letra);
          else
          {
            fraccion = fraccion * 10 + parseFloat(letra);
            divide *=10;
          }
        acumulabaNumero = true;
      }
      else
```

```
    {
      /* Si es un operador entonces crea el nodo del operador y el
      nodo del número si venía acumulando un número */
      if (acumulabaNumero)
      {
        objeto = new Pieza();
        objeto.ConstructorNumero(entero + fraccion/divide);
        this.Piezas[NumeroPieza++] = objeto;
      }

      /* Agrega a la lista */
      objeto = new Pieza();
      objeto.ConstructorOperador(letra);
      this.Piezas[NumeroPieza++] = objeto;

      /* Inicializa de nuevo las variables de conversión de string a número */
      entero = 0;
      fraccion = 0;
      divide = 1;
      puntoentero = false;
      acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    objeto = new Pieza();
    objeto.ConstructorNumero(entero + fraccion/divide);
    this.Piezas[NumeroPieza++] = objeto;
  }

};

/**
 * Ya construida la lista del tipo:
 * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
this.Evaluar = function()
{
  pos=0;
  antes=0;
  sigue=0;
  this.ERRORMATEMATICO = false;

  /* Total de nodos en la lista creada */
  totalPiezas = this.Piezas.length;

  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Activa todas las piezas para que sean evaluadas */
    this.Piezas[pos].setEvaluado(false);

    if (this.Piezas[pos].getTipo() == this.ESNUMERO)
      this.Piezas[pos].setAcumula(this.Piezas[pos].getNumero());
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es exponente */
    if (this.Piezas[pos].getTipo() == this.ESOPERADOR && this.Piezas[pos].getOperador() == '^')
    {
      /* Busca un nodo anterior que sea número y no haya sido evaluado */
      for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número y no haya sido evaluado */
      for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      this.Piezas[pos].setEvaluado(true);
      this.Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número elevado a Número */
      this.Piezas[antes].setAcumula(Math.pow(this.Piezas[antes].getAcumula(), this.Piezas[sigue].getAcumula()));
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es multiplicación o división */
    if (this.Piezas[pos].getTipo() == this.ESOPERADOR && (this.Piezas[pos].getOperador() == '*' ||
this.Piezas[pos].getOperador() == '/'))
    {
      /* Busca un nodo anterior que sea número y no haya sido evaluado */
      for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);
```

```
      /* Busca un nodo siguiente que sea número y no haya sido evaluado */
      for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      this.Piezas[pos].setEvaluado(true);
      this.Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número * Número, o Número / Número */
      if (this.Piezas[pos].getOperador() == '*')
        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() *  this.Piezas[sigue].getAcumula() );
      else
      {
        if (this.Piezas[sigue].getAcumula() != 0)
          this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() /  this.Piezas[sigue].getAcumula() );
        else
        {
          this.ERRORMATEMATICO = true;
          return 0;
        }
      }
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es suma o resta */
    if (this.Piezas[pos].getTipo() == this.ESOPERADOR && (this.Piezas[pos].getOperador() == '+' ||
this.Piezas[pos].getOperador() == '-'))
    {
      /* Busca un nodo anterior que sea número y no haya sido evaluado */
      for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número y no haya sido evaluado */
      for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      this.Piezas[pos].setEvaluado(true);
      this.Piezas[sigue].setEvaluado(true);

      /* Hace la operación de Número + Número, o Número - Número */
      if (this.Piezas[pos].getOperador() == '+')
        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() +  this.Piezas[sigue].getAcumula() );
      else
        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() -  this.Piezas[sigue].getAcumula() );
    }
   }

   /* Resultado de la expresión */
   return this.Piezas[0].getAcumula();
 };

 /**
  * Retorna true si hubo un error matemático
  * @return TRUE si hubo un error matemático
  */
 this.getERRORMATEMATICO = function()
 {
   return this.ERRORMATEMATICO;
 };
}
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

index.html

```
<html>
  <body>
    <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
      <script type="text/javascript">
      /* Probando el evaluador de expresiones */
      objEvaluador = new EvaluadorExpresionAlgebraica();
      document.write("Inicia");
      expresion = "7.795/8.941+0.242-9^3*6.609";
      objEvaluador.Analizar(expresion);
      valor = objEvaluador.Evaluar();
      document.write(valor);
      </script>
   </body>
</html>
```

Object Pascal

Pieza.pas

```
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web: http://darwin.50webs.com
// Licencia: LGPL
// Fecha: Enero de 2012
//
unit Pieza;

interface
type
  TPieza = class
  private
    // 1. Almacena un número, 2. Almacena un operador
    tipo: integer;

    // El número en la expresión algebraica */
    numero: double;

    // El operador (+, -, *, /, ^) en la expresión algebraica
    operador: char;

    //Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    //cálculos que determinan el valor de la expresión
    evaluado: boolean;

    //El resultado parcial de la evaluación de la expresión */
    acumula: double;

   public
     Constructor Create(numero: double); overload;
     Constructor Create(operador: char); overload;
     function getAcumula(): double;
     function isEvaluado(): boolean;
     function getNumero(): double;
     function getOperador(): char;
     function getTipo(): integer;
     procedure setAcumula(acumulado: double);
     procedure setEvaluado(evaluado: boolean);
   end;
implementation


// Constructor en caso que la pieza contenga un número real
// @param numero Número de tipo double
constructor TPieza.Create(numero: double);
begin
   self.tipo := 1; //Es un número
   self.numero := numero;

   self.evaluado := false;
   self.acumula := numero;
end;


// Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
// @param operador Que puede ser +, -, *, /, ^
constructor TPieza.Create(operador: char);
begin
    self.tipo := 2; //Es un operador
    self.operador := operador;

    self.evaluado := false;
    self.acumula := 0;
end;

// Retorna el acumulado que tiene esta pieza
// @return Acumulado de la pieza
function TPieza.getAcumula(): double;
begin
    getAcumula := acumula;
end;

// Retorna si la pieza ya fue evaluada
// @return True si la pieza ya fue evaluada
function TPieza.isEvaluado(): boolean;
begin
    isEvaluado := evaluado;
end;


// Retorna el número de tipo double que tiene la pieza
// @return El valor del número tipo double
function TPieza.getNumero(): double;
begin
    getNumero := numero;
end;

// Retorna el operador (+, -, *, /, ^) que tiene la pieza
```

```
// @return El operador en char
function TPieza.getOperador(): char;
begin
    getOperador := operador;
end;

// Retorna que tipo de pieza es: número u operador
// @return Tipo de pieza
function TPieza.getTipo(): integer;
begin
    getTipo := tipo;
end;

// Da valor al acumulado por pieza
// @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
procedure TPieza.setAcumula(acumulado:double);
begin
    self.acumula := acumulado;
end;

// Marca la pieza como ya evaluada
// @param evaluado true si la pieza ya ha sido evaluada, false si no
procedure TPieza.setEvaluado(evaluado: boolean);
begin
    self.evaluado := evaluado;
end;

end.
```

EvaluadorExpresionAlgebraica.pas

```
// Evaluación de una expresión algebraica simple:
// Sólo números reales y operadores (^, +, -, *, /)
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web:  http://darwin.50webs.com
// Licencia: LGPL
// Fecha:  Marzo de 2012
unit EvaluadorExpresionAlgebraica;

interface
uses
  //Requerido para el TobjectList
  Contnrs, Math, Pieza;

type
  TEvaluadorExpresionAlgebraica = class
  private
      //Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
      ESNUMERO: integer;
      ESOPERADOR: integer;

      //Esta constante sirve para que se reste al carácter y se obtenga el número
      //Ejemplo:  '7' - ASCIINUMERO =  7
      ASCIINUMERO: integer;

      //Lista simplemente enlazada que tiene los componentes (número u operador)
      //de la expresión algebraica ya dividida.
      Piezas: TobjectList;
      objPieza: TPieza;

      //Variable que se pone a true cuando hay un error matemático
      ERRORMATEMATICO: boolean;
  public
      Constructor Create();
      procedure Analizar(expr: string);
      function Evaluar(): double;
      function getErrorMatematico(): boolean;
      procedure LiberaMemoria();
  end;

implementation

constructor TEvaluadorExpresionAlgebraica.Create();
begin
   self.ESNUMERO := 1;
   self.ESOPERADOR := 2;
   self.ASCIINUMERO := 48;
end;

//Este método se encarga de analizar la expresión y convertirla en una
//estructura que permita evaluar la expresión.
//
//Convierte una expresión algebraica en una sucesión de nodos.
//|2| |+| |8.3| |/| |5| |*| |7.4|
//
//@param expr La expresión algebraica sin espacios y en minúsculas

procedure TEvaluadorExpresionAlgebraica.Analizar(expr: string);
var
  longitud: integer;
```

```
  //Va extrayendo letra a letra de la expresión simple */
  letra: char;

  //Conversión de string a double */
  entero, fraccion, divide: integer;
  puntoentero: boolean;

  //Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero: boolean;

  cont: integer;

begin
  // Inicializa la lista de piezas
  Piezas := TObjectList.Create;

  // Tamaño de la expresión simple
  longitud := length(expr);

  // Conversión de string a double
  entero := 0;
  fraccion := 0;
  divide := 1;
  puntoentero := false;

  // Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero := false;

for cont := 1 to longitud do
begin
  //Trae la letra
  letra := expr[cont];

  // Si es un número
  if (letra in ['0'..'9']) or (letra = '.') then
  begin
    //Ir armando el número de tipo double */
    if letra = '.' then
      puntoentero := true
    else
    begin
      if not(puntoentero) then // puntoentero = false
        entero := entero * 10 + ord(letra) - ASCIINUMERO
      else
      begin
        fraccion := fraccion * 10 + ord(letra) - ASCIINUMERO;
        divide := divide*10;
      end;
      acumulabaNumero := true;
    end
  end
  else
    begin
          // Si es un operador entonces crea el nodo del operador y el
          // nodo del número si venía acumulando un número */
          if acumulabaNumero then
          begin
            objPieza := TPieza.Create(entero + fraccion/divide);
            Piezas.Add(objPieza);
          end;

          // Agrega a la lista el operador */
          objPieza := TPieza.Create(letra);
          Piezas.Add(objPieza);

          // Inicializa de nuevo las variables de conversión de string a número */
          entero := 0;
          fraccion := 0;
          divide := 1;
          puntoentero := false;
          acumulabaNumero := false;
        end;
      end;

      // Cierra la expresión simple preguntando si el último operando es un número */
      if acumulabaNumero then
      begin
        objPieza := TPieza.Create(entero + fraccion/divide);
        Piezas.Add(objPieza);
      end;
    end;


// Ya construida la lista del tipo:
// [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
// es ir del operador con mas precedencia al de menos precedencia.
//
// Este método se llama después de haber sido analizada la expresión.
//
// @return El valor de la expresión evaluada (double)
function TEvaluadorExpresionAlgebraica.Evaluar(): double;
```

```
var
  pos, antes, sigue, totalPiezas: integer;
begin
  ERRORMATEMATICO := false;

  // Total de nodos en la lista creada
  totalPiezas := Piezas.Count - 1;

  for pos := 0 to totalPiezas do
  begin
    //Activa todas las piezas para que sean evaluadas */
    (Piezas.Items[pos] as TPieza).setEvaluado(false);

    if (Piezas.Items[pos] as Tpieza).getTipo() = ESNUMERO then
            (Piezas.Items[pos] as Tpieza).setAcumula((Piezas.Items[pos] as Tpieza).getNumero());
  end;

  // Recorre toda la lista
  for pos :=0 to  totalPiezas do
  begin
    //Si encuentra un nodo del tipo operador y es exponente */
    if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and ((Piezas.Items[pos] as Tpieza).getOperador() = '^') then
    begin
      //Busca un nodo anterior que sea número y no haya sido evaluado */
      antes := pos-1;
      while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

      //Busca un nodo siguiente que sea número y no haya sido evaluado */
      sigue := pos + 1;
      while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

      // Marca esos nodos actual y siguiente como ya evaluados */
      (Piezas.Items[pos] as Tpieza).setEvaluado(true);
      (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

       // Hace la operación de Número elevado a Número
       (Piezas.Items[antes] as Tpieza).setAcumula(Power((Piezas.Items[antes] as Tpieza).getAcumula(), (Piezas.Items[sigue] as
Tpieza).getAcumula()));
    end;
  end;

  // Recorre toda la lista
  for pos :=0 to  totalPiezas do
  begin
    // Si encuentra un nodo del tipo operador y es multiplicación o división */
    if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '*') or
((Piezas.Items[pos] as Tpieza).getOperador() = '/')) then
    begin
      //Busca un nodo anterior que sea número y no haya sido evaluado */
      antes := pos-1;
      while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

      //Busca un nodo siguiente que sea número y no haya sido evaluado */
      sigue := pos + 1;
      while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

      // Marca esos nodos actual y siguiente como ya evaluados */
      (Piezas.Items[pos] as Tpieza).setEvaluado(true);
      (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

      // Hace la operación de Número * Número, o Número / Número */
      if (Piezas.Items[pos] as Tpieza).getOperador() = '*' then
        (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() *  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
      else
      begin
        if (Piezas.Items[sigue] as Tpieza).getAcumula() <> 0 then
          (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() /  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
        else
        begin
          ERRORMATEMATICO := true;
          Evaluar := 0;
          Exit;
        end;
      end;
    end;
  end;

  // Recorre toda la lista
  for pos :=0 to  totalPiezas do
  begin
    // Si encuentra un nodo del tipo operador y es suma o resta */
    if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '+') or
((Piezas.Items[pos] as Tpieza).getOperador() = '-')) then
    begin
      //Busca un nodo anterior que sea número y no haya sido evaluado */
      antes := pos-1;
      while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

      //Busca un nodo siguiente que sea número y no haya sido evaluado */
      sigue := pos + 1;
```

```
      while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

      // Marca esos nodos actual y siguiente como ya evaluados */
      (Piezas.Items[pos] as Tpieza).setEvaluado(true);
      (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

      // Hace la operación de Número + Número, o Número - Número */
      if (Piezas.Items[pos] as Tpieza).getOperador() = '+' then
        (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() +  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
      else
        (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() -  (Piezas.Items[sigue] as
Tpieza).getAcumula() );
    end;
  end;

  // Resultado de la expresión
  Evaluar := (Piezas.Items[0] as Tpieza).getAcumula();
end;

// Retorna true si hubo un error matemático
// @return TRUE si hubo un error matemático
function TEvaluadorExpresionAlgebraica.getErrorMatematico(): boolean;
begin
  getErrorMatematico := ERRORMATEMATICO;
end;

//Libera la memoria
procedure TEvaluadorExpresionAlgebraica.LiberaMemoria();
begin
  Piezas.Free;
end;

end.
```

PARTE 3. INTERPRETANDO VARIABLES ALGEBRAICAS.

En una expresión algebraica se suelen usar variables, por ejemplo "4*x+5/y+z^2" la cual tiene las variables 'x', 'y', 'z'. En este evaluador de expresiones se podrá usar variables desde 'a' hasta 'z', es decir, 26 variables. El algoritmo de análisis de la expresión algebraica trabaja así:

1. Recorre la expresión algebraica y si encuentra una letra de 'a' a z' entonces genera una nueva pieza.
2. Esta nueva pieza se marca de tipo variable.

Cuando se está evaluando la expresión algebraica ya analizada, entonces el cambio es el siguiente:

1. El usuario previamente da valor a la variable con la instrucción. ¿Cómo? Llamando al método `DarValorVariable` el cual tiene dos parámetros: *variable* y *valor*, en el caso de *variable*, va desde 'a' hasta 'z' y en el *valor* es un valor real. Por ejemplo:

   ```
   objEvaluador.DarValorVariable('x', 17.9012);
   ```

   Ese método llena un arreglo unidimensional donde se almacena el valor de la variable

   ```
   /* Arreglo unidimensional con las 26 variables diferentes */
   double valorVariable[30];
   ```

   de ese modo, valorVariable[0] tendría el valor de la variable 'a', valorVariable[1] tendría el valor de la variable 'b', valorVariable[2] tendría el valor de la variable 'c' y así sucesivamente.

2. Cuando se recorre el arraylist de piezas y se detecta una pieza de tipo variable, entonces se toma el valor dado por el usuario.

```
for (pos=0; pos<totalPiezas; pos++)
{
  /* Activa todas las piezas para que sean evaluadas */
  Piezas[pos].setEvaluado(false);

  /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
    ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
  if (Piezas[pos].getTipo() == ESVARIABLE)
    Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
  else if (Piezas[pos].getTipo() == ESNUMERO)
    Piezas[pos].setAcumula(Piezas[pos].getNumero());
}
```

Un caso especial: La potencia

Si tenemos el caso de X^Y , se representa en el evaluador como X^Y. Si X es mayor de cero, no hay problema que suceda con el exponente, que puede ser positivo o negativo, real o entero. Ver ejemplos:

X	Y	X^Y (aproximación)
2	3	8
2	-3	8
2	0.4	1,3195
2	-0.4	0,7578
2	5.7	51,9841
2	-5.7	0,01923
2	0	1

Los problemas aparecen cuando la base es cero

X	Y	X^Y
0	2	0
0	-2	Infinito
0	0.4	0
0	-0.4	Infinito
0	5.7	0
0	-5.7	Infinito
0	0	1

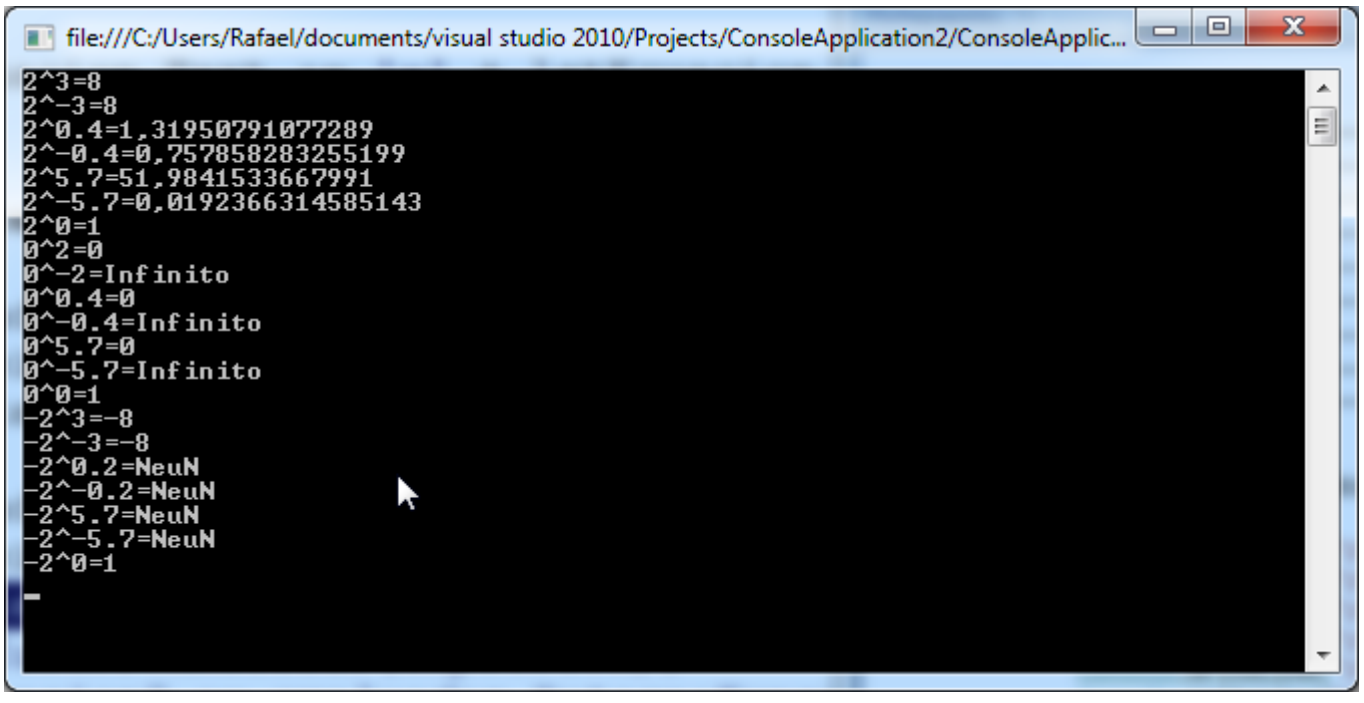
Y cuando la base es negativa

X	Y	X^Y
-2	3	-8
-2	-3	-8
-2	0.4	Error
-2	-0.4	Error
-2	5.7	Error
-2	-5.7	Error
-2	0	1

Se debe validar como error matemático tanto el “Infinito” como el “Error”.

Los diferentes casos con este código en C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("2^3=" + Math.Pow(2, 3));
            Console.WriteLine("2^-3=" + Math.Pow(2, 3));
            Console.WriteLine("2^0.4=" + Math.Pow(2, 0.4));
            Console.WriteLine("2^-0.4=" + Math.Pow(2, -0.4));
            Console.WriteLine("2^5.7=" + Math.Pow(2, 5.7));
            Console.WriteLine("2^-5.7=" + Math.Pow(2, -5.7));
            Console.WriteLine("2^0=" + Math.Pow(2, 0));

            Console.WriteLine("0^2=" + Math.Pow(0, 2));
            Console.WriteLine("0^-2=" + Math.Pow(0, -2));
            Console.WriteLine("0^0.4=" + Math.Pow(0, 0.4));
            Console.WriteLine("0^-0.4=" + Math.Pow(0, -0.4));
            Console.WriteLine("0^5.7=" + Math.Pow(0, 5.7));
            Console.WriteLine("0^-5.7=" + Math.Pow(0, -5.7));
            Console.WriteLine("0^0=" + Math.Pow(0, 0));

            Console.WriteLine("-2^3=" + Math.Pow(-2, 3));
            Console.WriteLine("-2^-3=" + Math.Pow(-2, 3));
            Console.WriteLine("-2^0.2=" + Math.Pow(-2, 0.2));
            Console.WriteLine("-2^-0.2=" + Math.Pow(-2, -0.2));
            Console.WriteLine("-2^5.7=" + Math.Pow(-2, 5.7));
            Console.WriteLine("-2^-5.7=" + Math.Pow(-2, -5.7));
            Console.WriteLine("-2^0=" + Math.Pow(-2, 0));

            Console.ReadKey();
        }
    }
}
```



La solución a este problema es hacer uso de las funciones de captura de errores de cada lenguaje de programación. Si se produce un error matemático, se captura y se pone la variable booleana de ERRORMATEMATICO a verdadero (true).

En JavaScript, por ejemplo, se retorna como resultado un NaN (not a number) cuando se produce un error matemático al ejecutar una potencia inválida. El error es capturado desde donde es utilizado el evaluador de expresiones. Ver el siguiente código:

```
      //Probando frente a un error matemático
      expresion = "2+x^0.4-3";
      objEvaluador.Analizar(expresion);
      objEvaluador.DarValorVariable('x', -2);
      valor = objEvaluador.Evaluar();
      if ( isNaN(valor) )
            document.write("Error matemático encontrado<br>");
```

En el caso de C++ se debe hacer uso de la variable global errno. Ver el siguiente código:

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
    char expresion[500];
    EvaluadorExpresionAlgebraica objEvaluador;
    double valor;

    strcpy(expresion, "1+x^0.4+3");
    objEvaluador.Analizar(expresion);
    objEvaluador.DarValorVariable('x', -2);
    errno=0; //Inicializa la variable global errno a cero
    valor=objEvaluador.Evaluar();
       if (errno != 0)
       printf("1. Error matematico capturado\n");
    else
       printf("%f\n", valor);

    objEvaluador.DarValorVariable('x', 2);
    errno=0;  //Inicializa la variable global errno a cero
    valor=objEvaluador.Evaluar();
    if (errno != 0)
       printf("2. Error matematico capturado\n");
    else
       printf("%f\n", valor);

    getchar();
    return 0;
}
```

Las piezas

Cada pieza tiene seis(6) atributos que son los siguientes:

- **Tipo:** Nos indica si la pieza almacena un número o un operador o una variable, de tipo entero.
- **Numero:** Almacena el número. De tipo de doble precisión.
- **Operador:** Almacena el operador (+, -, *, /, ^) . De tipo carácter.
- **Variable:** Almacena que variable es. De tipo entero.
- **Evaluado:** Si la pieza está marcada como evaluada tomará el valor true (verdadero). De tipo booleno.
- **Acumula:** Cuando se evalúa la expresión, como se vio en las figuras anteriores, el resultado de la operación se guarda en una pieza (reemplazando el valor que tenía). En la implementación, se guarda el nuevo valor en esta variable de tipo doble precisión porque de esa forma no se pierde la lista original y permite evaluar varias veces la expresión sin necesidad de volverla a construir. ¿Por qué evaluar varias veces? En el caso que la expresión tenga variables que cambian su valor en cada evaluación, un ejemplo, es si se desea hacer una gráfica de la ecuación.

La clase pieza tiene tres constructores:

Primer constructor: Llamado en caso que quiera almacenarse un número. Adicionalmente, copia ese número al atributo Acumula.

Segundo constructor: Llamado en caso que quiera almacenarse un operador

Tercer constructor: Llamado en caso que quiera almacenarse una variable

C++

Pieza.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

/* Pieza será la unidad básica de la expresión algebraica, cada nodo tendrá o un operador
   o un número o una variable. Este nodo trabaja en una lista simplemente enlazada */
class Pieza
{
private:
    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
    int tipo;

    /* El número en la expresión algebraica */
    double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    char operador;

    /* La variable */
    int variable;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    bool evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    double acumula;

public:
    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     */
    Pieza(double);

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     */
    Pieza(char);

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     */
    Pieza(int);

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
      double getAcumula();

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    bool isEvaluado();

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    double getNumero();

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    char getOperador();

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
     int getVariable();

    /**
     * Retorna que tipo de pieza es: número, operador o variable
     * @return Tipo de pieza
     */
    int getTipo();

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
```

```
    void setAcumula(double);

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(bool);
};
```

Pieza.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <stdio.h>
#include <math.h>
#include "Pieza.h"

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 */
Pieza::Pieza(double numero)
{
 this->tipo = 1; //Es un número
 this->numero = numero;

 this->evaluado = false;
 this->acumula = numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 */
Pieza::Pieza(char operador)
{
 this->tipo = 2; //Es un operador
 this->operador = operador;

 this->evaluado = false;
 this->acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 */
Pieza::Pieza(int variable)
{
 this->tipo = 3; //Es una variable
 this->variable = variable;

 this->evaluado = false;
 this->acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
double Pieza::getAcumula()
{
 return acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
bool Pieza::isEvaluado()
{
 return evaluado;
}

/**
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
double Pieza::getNumero()
{
 return numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
char Pieza::getOperador()
```

```
{
 return operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
int Pieza::getVariable()
{
 return variable;
}

/**
 * Retorna que tipo de pieza es: número, operador o variable
 * @return Tipo de pieza
 */
int Pieza::getTipo()
{
 return tipo;
}

/**
 * Da valor al acumulado por pieza
 * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
 */
void Pieza::setAcumula(double acumulado)
{
 this->acumula = acumulado;
}

/**
 * Marca la pieza como ya evaluada
 * @param evaluado true si la pieza ya ha sido evaluada, false si no
 */
void Pieza::setEvaluado(bool evaluado)
{
 this->evaluado = evaluado;
}
```

EvaluadorExpresionAlgebraica.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include "Pieza.h"
#include <vector>

class EvaluadorExpresionAlgebraica
{
private:
      /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    static const int ESNUMERO = 1;
    static const int ESOPERADOR = 2;
    static const int ESVARIABLE = 3;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    static const int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    static const int ASCIILETRA = 97;

    /* Lista simplemente enlazada que tiene los componentes (numero u operador
       de la expresión algebraica ya dividida */
    std::vector<Pieza> Piezas;

    /* Arreglo unidimensional con las 26 variables diferentes */
    double valorVariable[30];

      /* Variable que se pone a true cuando hay error matemático */
      bool ERRORMATEMATICO;

public:
    void DarValorVariable(char, double);

      /* Analiza la expresión algebraica y la vuelve una lista */
      void Analizar(char *);
```

```
      /* Evalúa la expresión que ha sido previamente analizada */
    double Evaluar(void);

      /* Retorna si hubo error matemático */
      bool getErrorMatematico();
};
```

EvaluadorExpresionAlgebraica.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <math.h>
#include <stdio.h>

#include "EvaluadorExpresionAlgebraica.h"

/**
 * Convierte una expresión simple en una sucesión de nodos.
 * |2| |+| |a| |/| |5| |*| |c| |-| |1.5|
*/
void EvaluadorExpresionAlgebraica::Analizar(char *expr)
{
  /* Inicializa la lista de piezas. http://www.codeguru.com/Cpp/Cpp/cpp_mfc/stl/article.php/c4027  */
  Piezas.clear();

  /* Conversión de string a double */
  int decimal = 0;
  int fraccion = 0;
  int divide = 1;
  bool puntodecimal = false;

  /* Tamaño de la expresión simple */
  int longitud = strlen(expr);

  /* Va extrayendo letra a letra de la expresión simple */
  char letra;

  /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
  bool acumulabaNumero = false;

  for (int cont=0; cont<longitud; cont++)
  {
    /* Trae la letra */
    letra = expr[cont];

    /* Si es una variable */
    if (letra >= 'a' && letra <= 'z')
    {
        Pieza objeto(letra - ASCIILETRA);
        Piezas.push_back(objeto);
    }
    /* Si es un número */
    else if( (letra >= '0' && letra <= '9') || letra == '.')
    {
      /* Ir armando el número de tipo double */
      if (letra == '.')
        puntodecimal = true;
      else
        if(!puntodecimal)  /* puntodecimal == false */
          decimal = decimal * 10 + letra-ASCIINUMERO;
        else
        {
          fraccion = fraccion * 10 + letra-ASCIINUMERO;
          divide *=10;
        }
      acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
         nodo del número si venía acumulando un número */
      if (acumulabaNumero)
      {
        /* El nodo del número */
        Pieza objeto(decimal + (double) fraccion/divide);

        /* Agrega a la lista */
        Piezas.push_back(objeto);
      }

      /* El nodo operador */
      Pieza objeto(letra);

      /* Agrega a la lista */
      Piezas.push_back(objeto);
```

```
      /* Inicializa de nuevo las variables de conversión de string a número */
      decimal = 0;
      fraccion = 0;
      divide = 1;
      puntodecimal = false;
      acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    Pieza objeto(decimal + (double) fraccion/divide);
    Piezas.push_back(objeto);
  }

}

/* Ya construida la lista del tipo:
   [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   es ir del operador con mas precedencia al de menos precedencia */
double EvaluadorExpresionAlgebraica::Evaluar()
{
  int pos=0, antes=0, sigue=0;
  ERRORMATEMATICO = false;

  /* Total de nodos en la lista creada */
  int totalPiezas = Piezas.size();

  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Activa todas las piezas para que sean evaluadas */
    Piezas[pos].setEvaluado(false);

    /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
      ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
    if (Piezas[pos].getTipo() == ESVARIABLE)
      Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
    else if (Piezas[pos].getTipo() == ESNUMERO)
      Piezas[pos].setAcumula(Piezas[pos].getNumero());
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
   /* Si encuentra un nodo del tipo operador y es exponente */
   if (Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
   {
    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
    for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
    for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

    /* Marca esos nodos actual y siguiente como ya evaluados */
    Piezas[pos].setEvaluado(true);
    Piezas[sigue].setEvaluado(true);

    /* Hace la operación de Número elevado a Número */
    Piezas[antes].setAcumula(pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
   }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
   /* Si encuentra un nodo del tipo operador y es multiplicación o división */
   if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' || Piezas[pos].getOperador() == '/'))
   {
    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
    for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
    for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

    /* Marca esos nodos actual y siguiente como ya evaluados */
    Piezas[pos].setEvaluado(true);
    Piezas[sigue].setEvaluado(true);

    /* Hace la operación de Número * Número, o Número / Número */
    if (Piezas[pos].getOperador() == '*')
     Piezas[antes].setAcumula( Piezas[antes].getAcumula() * Piezas[sigue].getAcumula() );
    else
    {
     if (Piezas[sigue].getAcumula() != 0)
      Piezas[antes].setAcumula( Piezas[antes].getAcumula() / Piezas[sigue].getAcumula() );
     else
     {
      ERRORMATEMATICO = true;
      return 0;
     }
```

```
    }
   }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
   /* Si encuentra un nodo del tipo operador y es suma o resta */
   if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' || Piezas[pos].getOperador() == '-'))
   {
    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
    for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
    for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

    /* Marca esos nodos actual y siguiente como ya evaluados */
    Piezas[pos].setEvaluado(true);
    Piezas[sigue].setEvaluado(true);

    /* Hace la operación de Número + Número, o Número - Número */
    if (Piezas[pos].getOperador() == '+')
     Piezas[antes].setAcumula( Piezas[antes].getAcumula() + Piezas[sigue].getAcumula() );
    else
     Piezas[antes].setAcumula( Piezas[antes].getAcumula() - Piezas[sigue].getAcumula() );
   }
  }

  /* Resultado de la expresión */
  return Piezas[0].getAcumula();
}

/* Guarda en un arreglo unidimensional el valor de las variables */
void EvaluadorExpresionAlgebraica::DarValorVariable(char variable, double valor)
{
   valorVariable[variable - ASCIILETRA] = valor;
}

/* Retorna true si hubo un error matemático */
bool EvaluadorExpresionAlgebraica::getErrorMatematico()
{
  return ERRORMATEMATICO;
}
```

Para probar este evaluador, se puede usar el siguiente código de ejemplo:

Principal.cpp

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
      char expresion[500];
      EvaluadorExpresionAlgebraica objEvaluador;
      double valor;
      strcpy(expresion, "5.418-0.899*x+0.795*4.742-8.782*3.794-x");
      objEvaluador.Analizar(expresion);
      objEvaluador.DarValorVariable('x', 17.9012);
      valor = objEvaluador.Evaluar();
      printf("%f\n", valor);
      getchar();
      return 0;
}
```

Para la captura de errores matemáticos, usemos este código de prueba:

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
    char expresion[500];
    EvaluadorExpresionAlgebraica objEvaluador;
    double valor;

    strcpy(expresion, "1+x^0.4+3");
    objEvaluador.Analizar(expresion);
    objEvaluador.DarValorVariable('x', -2);
    errno=0; //Inicializa la variable global errno a cero
    valor=objEvaluador.Evaluar();
    if (errno != 0)
       printf("1. Error matematico capturado\n");
    else
       printf("%f\n", valor);

    objEvaluador.DarValorVariable('x', 2);
    errno=0;  //Inicializa la variable global errno a cero
    valor=objEvaluador.Evaluar();
    if (errno != 0)
       printf("2. Error matematico capturado\n");
    else
       printf("%f\n", valor);

    getchar();
    return 0;
}
```

Visual C# .NET

Pieza.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    public class Pieza
    {
        /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
        private int tipo;

        /* El número en la expresión algebraica */
        private double numero;

        /* El operador (+, -, *, /, ^) en la expresión algebraica */
        private char operador;

        /* La variable */
        private int variable;

        /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
           cálculos que determinan el valor de la expresión */
        private bool evaluado;

        /* El resultado parcial de la evaluación de la expresión */
        private double acumula;

        /**
         * Constructor en caso que la pieza contenga un número real
         * @param numero Número de tipo double
         */
        public Pieza(double numero)
        {
            this.tipo = 1; //Es un número
            this.numero = numero;

            this.evaluado = false;
            this.acumula = numero;
        }

        /**
         * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
         * @param operador Que puede ser +, -, *, /, ^
         */
        public Pieza(char operador)
        {
            this.tipo = 2; //Es un operador
            this.operador = operador;

            this.evaluado = false;
            this.acumula = 0;
        }

        /**
         * Constructor en caso que el nodo contenga una variable
         * @param variable Puede ir de 0 a 25 que representa de a..z
         */
        public Pieza(int variable)
        {
            this.tipo = 3; //Es una variable
            this.variable = variable;

            this.evaluado = false;
            this.acumula = 0;
        }

        /** Retorna el acumulado que tiene esta pieza
         * @return Acumulado de la pieza
         */
        public double getAcumula()
        {
            return acumula;
        }

        /** Retorna si la pieza ya fue evaluada
         * @return True si la pieza ya fue evaluada
         */
        public bool isEvaluado()
        {
            return evaluado;
        }
```

```
        /**
         * Retorna el número de tipo double que tiene la pieza
         * @return El valor del número tipo double
         */
        public double getNumero()
        {
            return numero;
        }

        /**
         * Retorna el operador (+, -, *, /, ^) que tiene la pieza
         * @return El operador en char
         */
        public char getOperador()
        {
            return operador;
        }

        /**
         * Retorna la variable que tiene la pieza
         * @return La variable
         */
        public int getVariable()
        {
            return variable;
        }

        /**
         * Retorna que tipo de pieza es: número, operador, variable
         * @return Tipo de pieza
         */
        public int getTipo()
        {
            return tipo;
        }

        /**
         * Da valor al acumulado por pieza
         * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
         */
        public void setAcumula(double acumulado)
        {
            this.acumula = acumulado;
        }

         /**
         * Marca la pieza como ya evaluada
         * @param evaluado true si la pieza ya ha sido evaluada, false si no
         */
        public void setEvaluado(bool evaluado)
        {
            this.evaluado = evaluado;
        }
    }
}
```

EvaluadorExpresionAlgebraica.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
  class EvaluadorExpresionAlgebraica
  {
    /**
     * Constantes para determinar qué tipo es cada pieza en que se divide la expresión algebraica
     */
    private static int ESNUMERO = 1;
    private static int ESOPERADOR = 2;
    private static int ESVARIABLE = 3;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo: '7' - ASCIINUMERO = 7
     */
    private static int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo: 'b' - ASCIILETRA = 1
```

```
     */
    private static int ASCIILETRA = 97;

    /**
     * Lista simplemente enlazada que tiene los componentes (número, operador o variable)
     * de la expresión algebraica ya dividida.
     */
    private List<Pieza> Piezas = new List<Pieza>();

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    private double[] valorVariable = new double[26];

    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    private bool ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    public void Analizar(String expr)
    {
      /* Inicializa la lista de piezas */
      Piezas.Clear();

      /* Tamaño de la expresión simple */
      int longitud = expr.Length;

      /* Va extrayendo letra a letra de la expresión simple */
      char letra;

      /* Conversión de string a double */
      int entero = 0, fraccion = 0, divide = 1;
      bool puntoentero = false;

      /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
      bool acumulabaNumero = false;

      for (int cont=0; cont<longitud; cont++)
      {
        /* Trae la letra */
        letra = expr[cont];

        /* Si es una variable */
        if (letra >= 'a' && letra <= 'z')
        {
          /* Es una variable */
          Pieza objeto = new Pieza(letra - ASCIILETRA);
          Piezas.Add(objeto);
        }
        /* Si es un número */
        else if( (letra >= '0' && letra <= '9') || letra == '.')
        {
          /* Ir armando el número de tipo double */
          if (letra == '.')
            puntoentero = true;
          else
            if(!puntoentero) /* puntoentero == false */
              entero = entero * 10 + letra-ASCIINUMERO;
            else
            {
              fraccion = fraccion * 10 + letra-ASCIINUMERO;
              divide *=10;
            }
          acumulabaNumero = true;
        }
        else
        {
          /* Si es un operador entonces crea el nodo del operador y el
            nodo del número si venía acumulando un número */
          if (acumulabaNumero)
          {
            /* El nodo del número */
            Pieza objeto = new Pieza(entero + (double) fraccion/divide);

            /* Agrega a la lista */
            Piezas.Add(objeto);
          }

          /* El nodo operador */
          Pieza objeto2 = new Pieza(letra);

          /* Agrega a la lista */
          Piezas.Add(objeto2);
```

```
        /* Inicializa de nuevo las variables de conversión de string a número */
        entero = fraccion = 0;
        divide = 1;
        puntoentero = acumulabaNumero = false;
      }
    }

    /* Cierra la expresión simple preguntando si el último operando es un número */
    if (acumulabaNumero)
    {
      Pieza objeto = new Pieza(entero + (double) fraccion/divide);
      Piezas.Add(objeto);
    }
  }

  /**
   * Ya construida la lista del tipo:
   * [nodo número] [nodo operador] [nodo número] [nodo operador] .....
   * es ir del operador con más precedencia al de menos precedencia.
   *
   * Este método se llama después de haber sido analizada la expresión.
   *
   * En el caso que sólo cambia el valor de una variable, no es necesario
   * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
   * la misma expresión.
   *
   * @return El valor de la expresión evaluada (double)
   */
  public double Evaluar()
  {
    int pos=0, antes=0, sigue=0;
    ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    int totalPiezas = Piezas.Count;

    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Activa todas las piezas para que sean evaluadas */
      Piezas[pos].setEvaluado(false);

      /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
        ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
      if (Piezas[pos].getTipo() == ESVARIABLE)
        Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
      else if (Piezas[pos].getTipo() == ESNUMERO)
        Piezas[pos].setAcumula(Piezas[pos].getNumero());
    }


      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra un nodo del tipo operador y es exponente */
        if (Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
        {
          /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
          for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
          for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas[pos].setEvaluado(true);
          Piezas[sigue].setEvaluado(true);

          /* Hace la operación de Número elevado a Número */
          Piezas[antes].setAcumula(Math.Pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
        }
      }

      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra un nodo del tipo operador y es multiplicación o división */
        if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' || Piezas[pos].getOperador() == '/'))
        {
          /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
          for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
          for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas[pos].setEvaluado(true);
          Piezas[sigue].setEvaluado(true);

          /* Hace la operación de Número * Número, o Número / Número */
          if (Piezas[pos].getOperador() == '*')
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() * Piezas[sigue].getAcumula() );
          else
```

```
        {
          if (Piezas[sigue].getAcumula() != 0)
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() / Piezas[sigue].getAcumula() );
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
        }

      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es suma o resta */
      if (Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' || Piezas[pos].getOperador() == '-'))
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número + Número, o Número - Número */
        if (Piezas[pos].getOperador() == '+')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() + Piezas[sigue].getAcumula() );
        else
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() - Piezas[sigue].getAcumula() );
      }
    }

    /* Retorna el resultado */
    return Piezas[0].getAcumula();
  }

  /**
   * Guarda en un arreglo unidimensional el valor de las variables
   * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
   * @param valor Valor de tipo double que tendrá esa variable.
   */
  public void DarValorVariable(char variable, double valor)
  {
     valorVariable[variable - ASCIILETRA] = valor;
  }

  /**
   * Retorna true si hubo un error matemático
   * @return TRUE si hubo un error matemático
   */
  public bool getErrorMatematico()
  {
      return ERRORMATEMATICO;
  }
 }
}
```

Para probar el programa, usamos el siguiente código

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
            string expresion;
            double valor;

            expresion = "x*6.546/8.191*x+x";
            objEvaluador.Analizar(expresion);
            objEvaluador.DarValorVariable('x', 17.9012);
            valor = objEvaluador.Evaluar();
            Console.WriteLine(valor);
            Console.ReadKey();
        }
    }
}
```

Para la captura de errores, probemos con este código:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
           EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
           string expresion;
           double valor;
           expresion = "3+x^0.4-1";
           objEvaluador.Analizar(expresion);
           objEvaluador.DarValorVariable('x', -2);
           valor = objEvaluador.Evaluar();
           if (Double.IsNaN(valor))
                Console.WriteLine("Error matemático capturado");
           Console.ReadKey();
        }
    }
}
```

Visual Basic .NET

Pieza.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web:  http://darwin.50webs.com
'Licencia: LGPL
'Fecha:  Enero de 2012

Public Class Pieza
    '1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función
    Private tipo As Integer

    'El número en la expresión algebraica
    Private numero As Double

    'El operador (+, -, *, /, ^) en la expresión algebraica
    Private operador As Char

    'La variable
    Private variable As Integer

    'Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    'cálculos que determinan el valor de la expresión
    Private evaluado As Boolean

    'El resultado parcial de la evaluación de la expresión
    Private acumula As Double

    'Constructor en caso que la pieza contenga un número real
    '@param numero Número de tipo dim  as double
    '@param nivel Nivel en que se encuentra en la expresión este número
    Public Sub New(ByVal numero As Double)
        Me.tipo = 1 'Es un número
        Me.numero = numero

        Me.evaluado = False
        Me.acumula = numero
    End Sub

    'Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
    '@param operador Que puede ser +, -, *, /, ^
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal operador As Char)
        Me.tipo = 2 'Es un operador
        Me.operador = operador

        Me.evaluado = False
        Me.acumula = 0
    End Sub

    'Constructor en caso que el nodo contenga una variable
    '@param variable Puede ir de 0 a 25 que representa de a..z
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal variable As Integer)
        Me.tipo = 3 'Es una variable
        Me.variable = variable

        Me.evaluado = False
        Me.acumula = 0
    End Sub

    'Retorna el acumulado que tiene esta pieza
    '@return Acumulado de la pieza
    Public Function getAcumula() As Double
        Return acumula
    End Function

    'Retorna si la pieza ya fue evaluada
    '@return True si la pieza ya fue evaluada
    Public Function isEvaluado() As Boolean
        Return evaluado
    End Function

    'Retorna el número de tipo dim  as double que tiene la pieza
    '@return El valor del número tipo dim  as double
    Public Function getNumero() As Double
        Return numero
    End Function

    'Retorna el operador (+, -, *, /, ^) que tiene la pieza
    '@return El operador en dim  as char
    Public Function getOperador() As Char
        Return operador
    End Function

    'Retorna la variable que tiene la pieza
    '@return La variable
    Public Function getVariable() As Integer
        Return variable
    End Function
```

```
    'Retorna que tipo de pieza es: número, operador o variable
    '@return Tipo de pieza
    Public Function getTipo() As Integer
        Return tipo
    End Function


    'Da valor al acumulado por pieza
    '@param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
    Public Sub setAcumula(ByVal acumulado As Double)
        Me.acumula = acumulado
    End Sub

    'Marca la pieza como ya evaluada
    '@param evaluado true si la pieza ya ha sido evaluada, false si no
    Public Sub setEvaluado(ByVal evaluado As Boolean)
        Me.evaluado = evaluado
    End Sub

End Class
```

EvaluadorExpresionAlgebraica.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web:  http://darwin.50webs.com
'Licencia: LGPL
'Fecha:  Enero de 2012

Public Class EvaluadorExpresionAlgebraica
    'Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
    Private Shared ESNUMERO As Integer = 1
    Private Shared ESOPERADOR As Integer = 2
    Private Shared ESVARIABLE As Integer = 3

    'Esta constante sirve para que se reste al caracter y se obtenga el número
    'Ejemplo:  '7' - ASCIINUMERO =  7
    Private Shared ASCIINUMERO As Integer = 48

    'Esta constante sirve para que se reste al caracter y se obtenga el número
    'usado en el arreglo unidimensional que alberga los valores de las variables
    'Ejemplo:  'b' - ASCIILETRA =  1
    Private Shared ASCIILETRA As Integer = 97

    'Lista simplemente enlazada que tiene los componentes (número, operador o función)
    'de la expresión algebraica ya dividida.
    Private Piezas As New List(Of Pieza)()

    'Arreglo unidimensional con las 26 variables diferentes
    Private valorVariable As Double() = New Double(25) {}

    'Variable que se pone a true cuando hay un error matemático
    Private ERRORMATEMATICO As Boolean

    'Este método se encarga de analizar la expresión y convertirla en una
    'estructura que permita evaluar la expresión.
    '         *
    'Convierte una expresión algebraica en una sucesión de nodos.
    '|2| |+| |a| |/| |5| |*| |cos| |y|
    '|2| |+| |a| |/| |5| |*| |cos| |y|
    '
    '@param expr La expresión algebraica sin espacios y en minúsculas
    Public Sub Analizar(ByVal expr As [String])
        ' Inicializa la lista de piezas
        Piezas.Clear()

        ' Tamaño de la expresión simple
        Dim longitud As Integer = expr.Length

        ' Va extrayendo letra a letra de la expresión simple
        Dim letra As Char

        ' Conversión de string a double
        Dim entero As Integer = 0, fraccion As Integer = 0, divide As Integer = 1
        Dim puntoentero As Boolean = False

        ' Si llevaba acumulando un valor numérico esta variable se vuelve true
        Dim acumulabaNumero As Boolean = False

        For cont As Integer = 0 To longitud - 1
            ' Trae la letra
            letra = expr(cont)

            If letra >= "a" And letra <= "z" Then
                ' Es una variable
                Dim objeto As New Pieza(Asc(letra) - ASCIILETRA)
                Piezas.Add(objeto)
                ' Si es un número
            ElseIf (letra >= "0" And letra <= "9") Or letra = "." Then
                ' Ir armando el número de tipo double
                If letra = "." Then
                    puntoentero = True
```

```
            Else
                If Not puntoentero Then
                    ' puntoentero == false
                    entero = entero * 10 + Asc(letra) - ASCIINUMERO
                Else
                    fraccion = fraccion * 10 + Asc(letra) - ASCIINUMERO
                    divide *= 10
                End If
            End If
            acumulabaNumero = True
        Else
            ' Si es un operador entonces crea el nodo del operador y el
            ' nodo del número si venía acumulando un número
            If acumulabaNumero Then
                ' El nodo del número
                Dim objeto As New Pieza(entero + CDbl(fraccion) / divide)

                ' Agrega a la lista
                Piezas.Add(objeto)
            End If

            ' El nodo operador
            Dim objeto2 As New Pieza(letra)

            ' Agrega a la lista
            Piezas.Add(objeto2)

            ' Inicializa de nuevo las variables de conversión de string a número
            entero = 0
            fraccion = 0
            divide = 1
            puntoentero = False
            acumulabaNumero = False
        End If
    Next

    ' Cierra la expresión simple preguntando si el último operando es un número
    If acumulabaNumero Then
        Dim objeto As New Pieza(entero + CDbl(fraccion) / divide)
        Piezas.Add(objeto)
    End If
End Sub

'Ya construida la lista del tipo:
'[nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
'es ir del operador con más precedencia al de menos precedencia.
'
'Este método se llama después de haber sido analizada la expresión.
'
'En el caso que sólo cambia el valor de una variable, no es necesario
'analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
'la misma expresión.
'
'@return El valor de la expresión evaluada (double)
Public Function Evaluar() As Double
    Dim pos As Integer = 0, antes As Integer = 0, sigue As Integer = 0
    ERRORMATEMATICO = False

    ' Total de nodos en la lista creada
    Dim totalPiezas As Integer = Piezas.Count

    For pos = 0 To totalPiezas - 1
        ' Activa todas las piezas para que sean evaluadas
        Piezas(pos).setEvaluado(False)

        ' Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
        ' ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables.
        If Piezas(pos).getTipo() = ESVARIABLE Then
            Piezas(pos).setAcumula(valorVariable(Piezas(pos).getVariable()))
        ElseIf Piezas(pos).getTipo() = ESNUMERO Then
            Piezas(pos).setAcumula(Piezas(pos).getNumero())
        End If
    Next


    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es exponente
        If Piezas(pos).getTipo() = ESOPERADOR And Piezas(pos).getOperador() = "^" Then
            ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
            antes = pos - 1
            While Piezas(antes).isEvaluado()
                antes -= 1
            End While

            ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
            sigue = pos + 1
            While Piezas(sigue).isEvaluado()
                sigue += 1
            End While

            ' Marca esos nodos actual y siguiente como ya evaluados
```

```
            Piezas(pos).setEvaluado(True)
            Piezas(sigue).setEvaluado(True)

            ' Hace la operación de Número elevado a Número
            Piezas(antes).setAcumula(Math.Pow(Piezas(antes).getAcumula(), Piezas(sigue).getAcumula()))
        End If
    Next

    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es multiplicación o división
        If Piezas(pos).getTipo() = ESOPERADOR And (Piezas(pos).getOperador() = "*" Or Piezas(pos).getOperador() = "/") Then
            ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
            antes = pos - 1
            While Piezas(antes).isEvaluado()
                antes -= 1
            End While

            ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
            sigue = pos + 1
            While Piezas(sigue).isEvaluado()
                sigue += 1
            End While

            ' Marca esos nodos actual y siguiente como ya evaluados
            Piezas(pos).setEvaluado(True)
            Piezas(sigue).setEvaluado(True)

            ' Hace la operación de Número * Número, o Número / Número
            If Piezas(pos).getOperador() = "*" Then
                Piezas(antes).setAcumula(Piezas(antes).getAcumula() * Piezas(sigue).getAcumula())
            Else
                If Piezas(sigue).getAcumula() <> 0 Then
                    Piezas(antes).setAcumula(Piezas(antes).getAcumula() / Piezas(sigue).getAcumula())
                Else
                    ERRORMATEMATICO = True
                    Return 0
                End If

            End If
        End If
    Next

    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es suma o resta
        If Piezas(pos).getTipo() = ESOPERADOR And (Piezas(pos).getOperador() = "+" Or Piezas(pos).getOperador() = "-") Then
            ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
            antes = pos - 1
            While Piezas(antes).isEvaluado()
                antes -= 1
            End While

            ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
            sigue = pos + 1
            While Piezas(sigue).isEvaluado()
                sigue += 1
            End While

            ' Marca esos nodos actual y siguiente como ya evaluados
            Piezas(pos).setEvaluado(True)
            Piezas(sigue).setEvaluado(True)

            ' Hace la operación de Número + Número, o Número - Número
            If Piezas(pos).getOperador() = "+" Then
                Piezas(antes).setAcumula(Piezas(antes).getAcumula() + Piezas(sigue).getAcumula())
            Else
                Piezas(antes).setAcumula(Piezas(antes).getAcumula() - Piezas(sigue).getAcumula())
            End If
        End If
    Next

    ' Resultado de la evaluación
    Return Piezas(0).getAcumula()
End Function

'Guarda en un arreglo unidimensional el valor de las variables
'@param variable ¿Qué variable de la expresión recibirá el valor? a..z
'@param valor  Valor de tipo double que tendrá esa variable.
Public Sub DarValorVariable(ByVal variable As Char, ByVal valor As Double)
    valorVariable(Asc(variable) - ASCIILETRA) = valor
End Sub

'Retorna true si hubo un error matemático
'@return TRUE si hubo un error matemático
Public Function getErrorMatematico() As Boolean
    Return ERRORMATEMATICO
End Function

End Class
```

Module.vb

```
Module Module1

    Sub Main()
        Dim objEvaluador As New EvaluadorExpresionAlgebraica
        Dim valor, valorreal As Double
        Dim expresion As String

        expresion = "3.883/x+x-9.266*x+8.140+4.948+x/4.353+6.837*7.171-0.235"
        objEvaluador.Analizar(expresion)
        objEvaluador.DarValorVariable("x", 17.9012)
        valor = objEvaluador.Evaluar()
        Console.WriteLine(valor)
        Console.ReadKey()
    End Sub

End Module
```

Para capturar un error matemático podemos usar este código de prueba

```
Module Module1

    Sub Main()
        Dim objEvaluador As New EvaluadorExpresionAlgebraica
        Dim valor As Double
        Dim expresion As String

        expresion = "3+x^0.4-1"
        objEvaluador.Analizar(expresion)
        objEvaluador.DarValorVariable("x", -2)
        valor = objEvaluador.Evaluar()
        If Double.IsNaN(valor) Then
            Console.WriteLine("Error matemático capturado")
        End If
        Console.ReadKey()
    End Sub

End Module
```

Java

Pieza.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
public class Pieza
{
    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
    private int tipo;

    /* El número en la expresión algebraica */
    private double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    private char operador;

    /* La variable */
    private int variable;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    private boolean evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    private double acumula;

    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     * @param nivel Nivel en que se encuentra en la expresión este número
     */
    public Pieza(double numero)
    {
     this.tipo = 1; //Es un número
     this.numero = numero;

     this.evaluado = false;
     this.acumula = numero;
    }

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(char operador)
    {
     this.tipo = 2; //Es un operador
     this.operador = operador;

     this.evaluado = false;
     this.acumula = 0;
    }

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(int variable)
    {
     this.tipo = 3; //Es una variable
     this.variable = variable;

     this.evaluado = false;
     this.acumula = 0;
    }

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
    public double getAcumula()
    {
     return acumula;
    }

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    public boolean isEvaluado()
    {
     return evaluado;
    }

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
```

```
     */
    public double getNumero()
    {
     return numero;
    }

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    public char getOperador()
    {
     return operador;
    }

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    public int getVariable()
    {
     return variable;
    }

    /**
     * Retorna que tipo de pieza es: número, operador o variable
     * @return Tipo de pieza
     */
    public int getTipo()
    {
     return tipo;
    }

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    public void setAcumula(double acumulado)
    {
     this.acumula = acumulado;
    }

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(boolean evaluado)
    {
     this.evaluado = evaluado;
    }
}
```

EvaluadorExpresionAlgebraica.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
import java.util.ArrayList;

public class EvaluadorExpresionAlgebraica
{
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    private static final int ESNUMERO = 1;
    private static final int ESOPERADOR = 2;
    private static final int ESVARIABLE = 3;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    private static final int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    private static final int ASCIILETRA = 97;

    /**
     * Lista simplemente enlazada que tiene los componentes (número u operador)
     * de la expresión algebraica ya dividida.
     */
    private ArrayList<Pieza> Piezas = new ArrayList<Pieza>();

    /**
```

```
     * Arreglo unidimensional con las 26 variables diferentes
     */
    private double valorVariable[] = new double[26];


    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    private boolean ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    public void Analizar(String expr)
    {
        /* Inicializa la lista de piezas  */
        Piezas.clear();

        /* Tamaño de la expresión simple */
        int longitud = expr.length();

        /* Va extrayendo letra a letra de la expresión simple */
        char letra;

        /* Conversión de string a double */
        int decimal = 0, fraccion = 0, divide = 1;
        boolean puntodecimal = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        boolean acumulabaNumero = false;

        for (int cont=0; cont<longitud; cont++)
        {
            /* Trae la letra */
            letra = expr.charAt(cont);

            /* Si es una variable */
            if (letra >= 'a' && letra <= 'z')
            {
                Pieza objeto = new Pieza(letra - ASCIILETRA);
                Piezas.add(objeto);
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntodecimal = true;
                else
                    if(!puntodecimal)  /* puntodecimal == false */
                        decimal = decimal * 10 + letra-ASCIINUMERO;
                    else
                    {
                        fraccion = fraccion * 10 + letra-ASCIINUMERO;
                        divide *=10;
                    }
                    acumulabaNumero = true;
            }
            else
            {
                /* Si es un operador entonces crea el nodo del operador y el
                    nodo del número si venía acumulando un número */
                if (acumulabaNumero)
                {
                    /* El nodo del número */
                    Pieza objeto = new Pieza(decimal + (double) fraccion/divide);

                    /* Agrega a la lista */
                    Piezas.add(objeto);
                }

                /* El nodo operador */
                Pieza objeto = new Pieza(letra);

                /* Agrega a la lista */
                Piezas.add(objeto);

                /* Inicializa de nuevo las variables de conversión de string a número */
                decimal = fraccion = 0;
                divide = 1;
                puntodecimal = acumulabaNumero = false;
            }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
```

```
        {
            Pieza objeto = new Pieza(decimal + (double) fraccion/divide);
            Piezas.add(objeto);
        }
    }

    /**
     * Ya construida la lista del tipo:
     * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
     * es ir del operador con mas precedencia al de menos precedencia.
     *
     * Este método se llama después de haber sido analizada la expresión.
     *
     * En el caso que sólo cambia el valor de una variable, no es necesario
     * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
     * la misma expresión.
     *
     * @return El valor de la expresión evaluada (double)
     */
    public double Evaluar()
    {
        int pos=0, antes=0, sigue=0;
        ERRORMATEMATICO = false;

        /* Total de nodos en la lista creada */
        int totalPiezas = Piezas.size();

        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Activa todas las piezas para que sean evaluadas */
            Piezas.get(pos).setEvaluado(false);

            /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
               ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
            if (Piezas.get(pos).getTipo() == ESVARIABLE)
                Piezas.get(pos).setAcumula(valorVariable[Piezas.get(pos).getVariable()]);
            else if (Piezas.get(pos).getTipo() == ESNUMERO)
                Piezas.get(pos).setAcumula(Piezas.get(pos).getNumero());
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es exponente */
            if (Piezas.get(pos).getTipo() == ESOPERADOR && Piezas.get(pos).getOperador() == '^')
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                Piezas.get(pos).setEvaluado(true);
                Piezas.get(sigue).setEvaluado(true);

                /* Hace la operación de Número elevado a Número */
                Piezas.get(antes).setAcumula(Math.pow(Piezas.get(antes).getAcumula(), Piezas.get(sigue).getAcumula()));
            }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es multiplicación o división */
            if (Piezas.get(pos).getTipo() == ESOPERADOR && (Piezas.get(pos).getOperador() == '*' ||
Piezas.get(pos).getOperador() == '/'))
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                Piezas.get(pos).setEvaluado(true);
                Piezas.get(sigue).setEvaluado(true);

                /* Hace la operación de Número * Número, o Número / Número */
                if (Piezas.get(pos).getOperador() == '*')
                    Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() *  Piezas.get(sigue).getAcumula() );
                else
                {
                    if (Piezas.get(sigue).getAcumula() != 0)
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() /  Piezas.get(sigue).getAcumula() );
                    else
                    {
                        ERRORMATEMATICO = true;
                        return 0;
                    }
                }
            }
```

```
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
                /* Si encuentra un nodo del tipo operador y es suma o resta */
                if (Piezas.get(pos).getTipo() == ESOPERADOR && (Piezas.get(pos).getOperador() == '+' ||
Piezas.get(pos).getOperador() == '-'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    Piezas.get(pos).setEvaluado(true);
                    Piezas.get(sigue).setEvaluado(true);

                    /* Hace la operación de Número + Número, o Número - Número */
                    if (Piezas.get(pos).getOperador() == '+')
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() +  Piezas.get(sigue).getAcumula() );
                    else
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() -  Piezas.get(sigue).getAcumula() );
                }
            }

        /* Resultado de la expresión */
        return Piezas.get(0).getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    public void DarValorVariable(char variable, double valor)
    {
         valorVariable[variable - ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    public boolean getErrorMatematico()
    {
            return ERRORMATEMATICO;
    }
}
```

Para probar el evaluador, podemos usar el siguiente código:

Inicio.java

```
public class Inicio
{
    public static void main(String args[])
    {
        EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
        String expresion;
        double valor;

        expresion = "5.38+9.76/x+x";
        objEvaluador.Analizar(expresion);
        objEvaluador.DarValorVariable('x', 17.9012);
        valor = objEvaluador.Evaluar();
        System.out.println(valor);
    }
}
```

Para captura de error, tenemos este código de ejemplo

```
public class CapturaError
{
    public static void main(String args[])
    {
        EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
        String expresion;
        Double valor;

        expresion = "3-x^0.4+1";
        objEvaluador.Analizar(expresion);
        objEvaluador.DarValorVariable('x', -2);
        valor = objEvaluador.Evaluar();
        if (valor.isNaN())
           System.out.println("Error matemático capturado");
        else
           System.out.println(valor);
    }
}
```

PHP

Pieza.php

```php
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
class Pieza
{
  /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
  var $nivel;

  /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
  var $tipo;

  /* El número en la expresión algebraica */
  var $numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  var $operador;

  /* La variable */
  var $variable;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
var $evaluado;

/* El resultado parcial de la evaluación de la expresión */
var $acumula;

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 */
public function ConstructorNumero($numero)
{
 $this->tipo = 1; //Es un número
 $this->numero = $numero;

 $this->evaluado = false;
 $this->acumula = $numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 */
public function ConstructorOperador($operador)
{
  $this->tipo = 2; //Es un operador
  $this->operador = $operador;

  $this->evaluado = false;
  $this->acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
public function ConstructorVariable($variable)
{
  $this->tipo = 3; //Es una variable
  $this->variable = $variable;

  $this->evaluado = false;
  $this->acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
public function getAcumula()
{
  return $this->acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
function isEvaluado()
{
  return $this->evaluado;
}

/**
```

```
  * Retorna el número de tipo double que tiene la pieza
  * @return El valor del número tipo double
  */
 public function getNumero()
 {
   return $this->numero;
 }

 /**
  * Retorna el operador (+, -, *, /, ^) que tiene la pieza
  * @return El operador en char
  */
 public function getOperador()
 {
   return $this->operador;
 }

 /**
  * Retorna la variable que tiene la pieza
  * @return La variable
  */
 public function getVariable()
 {
  return $this->variable;
 }

 /**
  * Retorna que tipo de pieza es: número u operador
  * @return Tipo de pieza
  */
  public function getTipo()
  {
    return $this->tipo;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  public function setAcumula($acumulado)
  {
    $this->acumula = $acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  public function setEvaluado($evaluado)
  {
    $this->evaluado = $evaluado;
  }
}
?>
```

EvaluadorExpresionAlgebraica.php

```
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
include("Pieza.php");

class EvaluadorExpresionAlgebraica
{
 /**
  * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
  */
 var $ESNUMERO = 1;
 var $ESOPERADOR = 2;
 var $ESVARIABLE = 3;

 /**
  * Esta constante sirve para que se reste al carácter y se obtenga el número
  * Ejemplo: '7' - ASCIINUMERO = 7
  */
 var $ASCIINUMERO = 48;

 /**
  * Esta constante sirve para que se reste al carácter y se obtenga el número
  * usado en el arreglo unidimensional que alberga los valores de las variables
  * Ejemplo: 'b' - ASCIILETRA = 1
  */
 var $ASCIILETRA = 97;

 /**
  * Lista simplemente enlazada que tiene los componentes (número u operador)
```

```
 * de la expresión algebraica ya dividida.
 */
var $Piezas = array();

/**
 * Arreglo unidimensional con las 26 variables diferentes
 */
var $valorVariable = array();

/**
 * Variable que se pone a true cuando hay un error matemático
 */
var $ERRORMATEMATICO;

/**
 * Este método se encarga de analizar la expresión y convertirla en una
 * estructura que permita evaluar la expresión.
 *
 * Convierte una expresión algebraica en una sucesión de nodos.
 * |2| |+| |8.3| |/| |5| |*| |7.4|
 *
 * @param expr La expresión algebraica sin espacios y en minúsculas
*/
public function Analizar($expr)
{
 /* Inicializa la lista de piezas */
 unset($this->Piezas);

 /* Tamaño de la expresión simple */
$longitud = strlen($expr);

/* Conversión de string a double */
$entero = 0;
$fraccion = 0;
$divide = 1;
$puntoentero = false;

/* Si llevaba acumulando un valor numérico esta variable se vuelve true */
$acumulabaNumero = false;

for ($cont=0; $cont<$longitud; $cont++)
{
 /* Trae la letra */
 $letra = $expr{$cont};

 /* Si es una variable */
 if ($letra >= 'a' && $letra <= 'z')
 {
   $objeto = new Pieza();
   $objeto->ConstructorVariable(ord($letra) - $this->ASCIILETRA);
   $this->Piezas[] = $objeto;
 }
 /* Si es un número */
 else if( ($letra >= '0' && $letra <= '9') || $letra == '.')
 {
  /* Ir armando el número de tipo double */
  if ($letra == '.')
   $puntoentero = true;
  else
   if(!$puntoentero) /* $puntoentero == false */
 $entero = $entero * 10 + $letra;
   else
   {
 $fraccion = $fraccion * 10 + $letra;
 $divide *=10;
   }
   $acumulabaNumero = true;
 }
 else
 {
  /* Si es un operador entonces crea el nodo del operador y el
  nodo del número si venía acumulando un número */
  if ($acumulabaNumero)
  {
   $objeto = new Pieza();
   $objeto->ConstructorNumero($entero + $fraccion/$divide);
   $this->Piezas[] = $objeto;
  }

  /* Agrega a la lista */
  $objeto = new Pieza();
  $objeto->ConstructorOperador($letra);
  $this->Piezas[] = $objeto;

  /* Inicializa de nuevo las variables de conversión de string a número */
  $entero = 0;
  $fraccion = 0;
  $divide = 1;
  $puntoentero = false;
  $acumulabaNumero = false;
 }
}
```

```
 /* Cierra la expresión simple preguntando si el último operando es un número */
 if ($acumulabaNumero)
 {
  $objeto = new Pieza();
  $objeto->ConstructorNumero($entero + $fraccion/$divide);
  $this->Piezas[] = $objeto;
 }
}

/**
 * Ya construida la lista del tipo:
 * [nodo número] [nodo operador] [nodo número] [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
  public function Evaluar()
  {
     $pos=0;
     $antes=0;
     $sigue=0;
     $this->ERRORMATEMATICO = false;

     /* Total de nodos en la lista creada */
     $totalPiezas = sizeof($this->Piezas);

    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
       /* Activa todas las piezas para que sean evaluadas */
       $this->Piezas[$pos]->setEvaluado(false);

       /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
       ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
       if ($this->Piezas[$pos]->getTipo() == $this->ESVARIABLE)
          $this->Piezas[$pos]->setAcumula($this->valorVariable[$this->Piezas[$pos]->getVariable()]);
       else if ($this->Piezas[$pos]->getTipo() == $this->ESNUMERO)
          $this->Piezas[$pos]->setAcumula($this->Piezas[$pos]->getNumero());
    }

    /* Recorre toda la lista */
    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
     /* Si encuentra un nodo del tipo operador y es exponente */
     if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && $this->Piezas[$pos]->getOperador() == '^')
     {
       /* Busca un nodo anterior que sea número y no haya sido evaluado */
       for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

       /* Busca un nodo siguiente que sea número y no haya sido evaluado */
       for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

       /* Marca esos nodos actual y siguiente como ya evaluados */
       $this->Piezas[$pos]->setEvaluado(true);
       $this->Piezas[$sigue]->setEvaluado(true);

       /* Hace la operación de Número elevado a Número */
       $this->Piezas[$antes]->setAcumula(pow($this->Piezas[$antes]->getAcumula(), $this->Piezas[$sigue]->getAcumula()));
     }
    }

    /* Recorre toda la lista */
    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
     /* Si encuentra un nodo del tipo operador y es multiplicación o división */
     if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this->Piezas[$pos]->getOperador() == '*' || $this-
>Piezas[$pos]->getOperador() == '/'))
     {
       /* Busca un nodo anterior que sea número y no haya sido evaluado */
       for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

       /* Busca un nodo siguiente que sea número y no haya sido evaluado */
       for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

       /* Marca esos nodos actual y siguiente como ya evaluados */
       $this->Piezas[$pos]->setEvaluado(true);
       $this->Piezas[$sigue]->setEvaluado(true);

       /* Hace la operación de Número * Número, o Número / Número */
       if ($this->Piezas[$pos]->getOperador() == '*')
        $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() * $this->Piezas[$sigue]->getAcumula() );
       else
       {
        if ($this->Piezas[$sigue]->getAcumula() != 0)
          $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() / $this->Piezas[$sigue]->getAcumula() );
        else
        {
          $this->ERRORMATEMATICO = true;
          return 0;
        }
```

```
          }
        }
      }

      /* Recorre toda la lista */
      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
         /* Si encuentra un nodo del tipo operador y es suma o resta */
         if ($this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this->Piezas[$pos]->getOperador() == '+' || $this-
>Piezas[$pos]->getOperador() == '-'))
         {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          $this->Piezas[$pos]->setEvaluado(true);
          $this->Piezas[$sigue]->setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if ($this->Piezas[$pos]->getOperador() == '+')
            $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() + $this->Piezas[$sigue]->getAcumula() );
          else
            $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() - $this->Piezas[$sigue]->getAcumula() );
         }
      }

      /* Resultado de la expresión */
      return $this->Piezas[0]->getAcumula();
    }

 /**
  * Guarda en un arreglo unidimensional el valor de las variables
  * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
  * @param valor Valor de tipo double que tendrá esa variable.
  */
 public function DarValorVariable($variable, $valor)
 {
   $this->valorVariable[ord($variable) - $this->ASCIILETRA] = $valor;
 }

 /**
  * Retorna true si hubo un error matemático
  * @return TRUE si hubo un error matemático
  */
 public function getERRORMATEMATICO()
 {
    return $this->ERRORMATEMATICO;
 }
}
?>
```

Para probar el evaluador, usamos el siguiente código

Index.php

```
<?php
/*
   Software:
   Licencia: GNU/GPL
   Autor: Ing. Rafael Alberto Moreno Parra (ramsoftware@gmail.com). Página Web: http://darwin.50webs.com
   Fecha: Enero de 2012
*/
include("EvaluadorExpresionAlgebraica.php");

$objEvaluador = new EvaluadorExpresionAlgebraica();

echo "Inicia<br>";

$expresion = '5.9/x*0.308+8.34/x+1';
$objEvaluador->Analizar($expresion);
$objEvaluador->DarValorVariable('x', 17.9012);
$valor = $objEvaluador->Evaluar();
echo $valor;
?>
```

Para ver la captura de errores matemáticos, probamos con este código:

```php
<?php
/*
   Software:
   Licencia: GNU/GPL
   Autor: Ing. Rafael Alberto Moreno Parra (ramsoftware@gmail.com). Página Web: http://darwin.50webs.com
   Fecha: Enero de 2012
*/
include("EvaluadorExpresionAlgebraica.php");

$objEvaluador = new EvaluadorExpresionAlgebraica();

echo "Inicia<br>";

$expresion = '3-x^0.4+1';
$objEvaluador->Analizar($expresion);
$objEvaluador->DarValorVariable('x', -2);
$valor = $objEvaluador->Evaluar();
if (is_nan($valor))
      echo "Error matemático detectado";
else
      echo $valor;
?>
```

JAVASCRIPT

EvaluadorExpresionAlgebraica.js

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
function Pieza()
{
  /* 1. Almacena un número, 2. Almacena un operador */
  this.tipo;

  /* El número en la expresión algebraica */
  this.numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  this.operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  this.evaluado;

  /* El resultado parcial de la evaluación de la expresión */
  this.acumula;

  /**
   * Constructor en caso que la pieza contenga un número real
   * @param numero Número de tipo double
   */
  this.ConstructorNumero = function(numero)
  {
   this.tipo = 1; //Es un número
   this.numero = numero;

   this.evaluado = false;
   this.acumula = numero;
  }

  /**
   * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
   * @param operador Que puede ser +, -, *, /, ^
   */
  this.ConstructorOperador = function(operador)
  {
    this.tipo = 2; //Es un operador
    this.operador = operador;

    this.evaluado = false;
    this.acumula = 0;
  }

  /**
   * Constructor en caso que el nodo contenga una variable
   * @param variable Puede ir de 0 a 25 que representa de a..z
   */
  this.ConstructorVariable = function(variable)
  {
   this.tipo = 3; //Es una variable
   this.variable = variable;

   this.evaluado = false;
   this.acumula = 0;
  }

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
  this.getAcumula = function()
  {
    return this.acumula;
  }

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
  this.isEvaluado = function()
  {
    return this.evaluado;
  }

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
  this.getNumero = function()
  {
    return this.numero;
```

```
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
this.getOperador = function()
{
  return this.operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
this.getVariable = function()
{
  return this.variable;
}

/**
 * Retorna que tipo de pieza es: número u operador
 * @return Tipo de pieza
 */
this.getTipo = function()
{
  return this.tipo;
}

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  this.setAcumula = function(acumulado)
  {
    this.acumula = acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  this.setEvaluado = function(evaluado)
  {
    this.evaluado = evaluado;
  }
}

/**
  ***********************************************************************************************************

  ***********************************************************************************************************
*/


function EvaluadorExpresionAlgebraica()
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
{
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    this.ESNUMERO = 1;
    this.ESOPERADOR = 2;
    this.ESVARIABLE = 3;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    this.ASCIILETRA = 97;

    /**
     * Lista simplemente enlazada que tiene los componentes (número, operador o función)
     * de la expresión algebraica ya dividida.
     */
    this.Piezas = [];

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    this.valorVariable = [];

    /**
     * Variable que se pone a true cuando hay un error matemático
```

```
     */
    this.ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    this.Analizar = function(expr)
    {
        /* Inicializa la lista de piezas  */
        this.Piezas.length = 0;
        NumeroPieza = 0;

        /* Tamaño de la expresión simple */
        longitud = expr.length;

        /* Conversión de string a double */
        entero = 0;
        fraccion = 0;
        divide = 1;
        puntoentero = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        acumulabaNumero = false;

        for (cont=0; cont<longitud; cont++)
        {
            /* Trae la letra */
            letra = expr.charAt(cont);

            /* Si es una variable o una función */
            if (letra >= 'a' && letra <= 'z')
            {
                    objeto = new Pieza();
                    objeto.ConstructorVariable(letra - this.ASCIILETRA);
                    this.Piezas[NumeroPieza++] = objeto;
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntoentero = true;
                else
                    if(!puntoentero)  /* puntoentero == false */
                        entero = entero * 10 + parseFloat(letra);
                    else
                    {
                        fraccion = fraccion * 10 + parseFloat(letra);
                        divide *=10;
                    }
                    acumulabaNumero = true;
            }
            else
            {
                /* Si es un operador entonces crea el nodo del operador y el
                    nodo del número si venía acumulando un número */
                if (acumulabaNumero)
                {
                    /* El nodo del número */
                    objeto = new Pieza();
                    objeto.ConstructorNumero(entero + fraccion/divide);
                    this.Piezas[NumeroPieza++] = objeto;
                }

                /* El nodo operador */
                objeto = new Pieza();
                objeto.ConstructorOperador(letra);

                /* Agrega a la lista */
                this.Piezas[NumeroPieza++] = objeto;

                /* Inicializa de nuevo las variables de conversión de string a número */
                entero = fraccion = 0;
                divide = 1;
                puntoentero = acumulabaNumero = false;
            }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
        {
            objeto = new Pieza();
            objeto.ConstructorNumero(entero + fraccion/divide);
            this.Piezas[NumeroPieza++] = objeto;
        }
    }
```

```
/**
 * Ya construida la lista del tipo:
 * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * En el caso que sólo cambia el valor de una variable, no es necesario
 * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
 * la misma expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
this.Evaluar = function()
{
    pos=0, antes=0, sigue=0;
    ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    totalPiezas = this.Piezas.length;

    for (pos=0; pos<totalPiezas; pos++)
    {
        /* Activa todas las piezas para que sean evaluadas */
        this.Piezas[pos].setEvaluado(false);

        /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
           ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
        if (this.Piezas[pos].getTipo() == this.ESVARIABLE)
            this.Piezas[pos].setAcumula(this.valorVariable[this.Piezas[pos].getVariable()]);
        else if (this.Piezas[pos].getTipo() == this.ESNUMERO)
            this.Piezas[pos].setAcumula(this.Piezas[pos].getNumero());
    }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es exponente */
            if (this.Piezas[pos].getTipo() == this.ESOPERADOR && this.Piezas[pos].getOperador() == '^')
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                this.Piezas[pos].setEvaluado(true);
                this.Piezas[sigue].setEvaluado(true);

                /* Hace la operación de Número elevado a Número */
                this.Piezas[antes].setAcumula(Math.pow(this.Piezas[antes].getAcumula(), this.Piezas[sigue].getAcumula()));
            }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es multiplicación o división */
            if (this.Piezas[pos].getTipo() == this.ESOPERADOR && (this.Piezas[pos].getOperador() == '*' ||
this.Piezas[pos].getOperador() == '/'))
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                this.Piezas[pos].setEvaluado(true);
                this.Piezas[sigue].setEvaluado(true);

                /* Hace la operación de Número * Número, o Número / Número */
                if (this.Piezas[pos].getOperador() == '*')
                    this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() *  this.Piezas[sigue].getAcumula() );
                else
                {
                    if (this.Piezas[sigue].getAcumula() != 0)
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() /  this.Piezas[sigue].getAcumula() );
                    else
                    {
                        ERRORMATEMATICO = true;
                        return 0;
                    }
                }

            }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
```

```
            {
                /* Si encuentra un nodo del tipo operador y es suma o resta */
                if (this.Piezas[pos].getTipo() == this.ESOPERADOR && (this.Piezas[pos].getOperador() == '+' ||
this.Piezas[pos].getOperador() == '-'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    this.Piezas[pos].setEvaluado(true);
                    this.Piezas[sigue].setEvaluado(true);

                    /* Hace la operación de Número + Número, o Número - Número */
                    if (this.Piezas[pos].getOperador() == '+')
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() +  this.Piezas[sigue].getAcumula() );
                    else
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() -  this.Piezas[sigue].getAcumula() );
                }
            }

        /* Resultado de la expresión */
        return this.Piezas[0].getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    this.DarValorVariable = function(variable, valor)
    {
         this.valorVariable[variable - this.ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    this.getErrorMatematico = function()
    {
            return ERRORMATEMATICO;
    }
}
```

Para probar el evaluador, use este código en un archivo HTML

index.html

```
<html>
  <body>
    <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
      <script type="text/javascript">
      /* Probando el evaluador de expresiones */
      objEvaluador = new EvaluadorExpresionAlgebraica();
      expresion = "x-6.4*8.32-x+x/5.545*2.110+6.193*6.659-x";
      objEvaluador.Analizar(expresion);
      objEvaluador.DarValorVariable('x', 17.9012);
      valor = objEvaluador.Evaluar();
      document.write(valor);
      </script>
   </body>
</html>
```

Para probar el evaluador, en captura de errores, use este código en un archivo HTML

```
<html>
  <body>
    <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
      <script type="text/javascript">
      /* Probando el evaluador de expresiones */
      objEvaluador = new EvaluadorExpresionAlgebraica();

      //Probando un error matemático
      expresion = "0/0";
      objEvaluador.Analizar(expresion);
      valor = objEvaluador.Evaluar();
      if (objEvaluador.getErrorMatematico() == true)
            document.write("Error matemático encontrado: 0/0<br>");

      //Probando un error matemático
      expresion = "2+x^0.4-3";
      objEvaluador.Analizar(expresion);
      objEvaluador.DarValorVariable('x', -2);
      valor = objEvaluador.Evaluar();
      if ( isNaN(valor) )
            document.write("Error matemático encontrado: -2^0.4<br>");
      </script>
   </body>
</html>
```

Object Pascal

Pieza.pas

```
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web: http://darwin.50webs.com
// Licencia: LGPL
// Fecha: Enero de 2012
//
unit Pieza;

interface
type
  TPieza = class
  private
    // 1. Almacena un número, 2. Almacena un operador
    tipo: integer;

    // El número en la expresión algebraica */
    numero: double;

    // El operador (+, -, *, /, ^) en la expresión algebraica
    operador: char;

    // La variable
    variable: integer;

    //Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    //cálculos que determinan el valor de la expresión
    evaluado: boolean;

    //El resultado parcial de la evaluación de la expresión */
    acumula: double;

   public
     Constructor Create(numero: double); overload;
     Constructor Create(operador: char); overload;
     Constructor Create(variable: integer); overload;
     function getAcumula(): double;
     function isEvaluado(): boolean;
     function getNumero(): double;
     function getOperador(): char;
     function getVariable(): integer;
     function getTipo(): integer;
     procedure setAcumula(acumulado: double);
     procedure setEvaluado(evaluado: boolean);
   end;
implementation


// Constructor en caso que la pieza contenga un número real
// @param numero Número de tipo double
constructor TPieza.Create(numero: double);
begin
   self.tipo := 1; //Es un número
   self.numero := numero;

   self.evaluado := false;
   self.acumula := numero;
end;


// Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
// @param operador Que puede ser +, -, *, /, ^
constructor TPieza.Create(operador: char);
begin
    self.tipo := 2; //Es un operador
    self.operador := operador;

    self.evaluado := false;
    self.acumula := 0;
end;


// Constructor en caso que el nodo contenga una variable
// @param variable Puede ir de 0 a 25 que representa de a..z
constructor TPieza.Create(variable: integer);
begin
     self.tipo := 3; //Es una variable
     self.variable := variable;

     self.evaluado := false;
     self.acumula := 0;
end;

// Retorna el acumulado que tiene esta pieza
// @return Acumulado de la pieza
function TPieza.getAcumula(): double;
begin
```

```
    getAcumula := acumula;
end;

// Retorna si la pieza ya fue evaluada
// @return True si la pieza ya fue evaluada
function TPieza.isEvaluado(): boolean;
begin
    isEvaluado := evaluado;
end;


// Retorna el número de tipo double que tiene la pieza
// @return El valor del número tipo double
function TPieza.getNumero(): double;
begin
    getNumero := numero;
end;

// Retorna el operador (+, -, *, /, ^) que tiene la pieza
// @return El operador en char
function TPieza.getOperador(): char;
begin
    getOperador := operador;
end;

// Retorna la variable que tiene la pieza
// @return La variable
function TPieza.getVariable(): integer;
begin
    getVariable := variable;
end;

// Retorna que tipo de pieza es: número u operador
// @return Tipo de pieza
function TPieza.getTipo(): integer;
begin
    getTipo := tipo;
end;

// Da valor al acumulado por pieza
// @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
procedure TPieza.setAcumula(acumulado:double);
begin
    self.acumula := acumulado;
end;

// Marca la pieza como ya evaluada
// @param evaluado true si la pieza ya ha sido evaluada, false si no
procedure TPieza.setEvaluado(evaluado: boolean);
begin
    self.evaluado := evaluado;
end;

end.
```

EvaluadorExpresionAlgebraica.pas

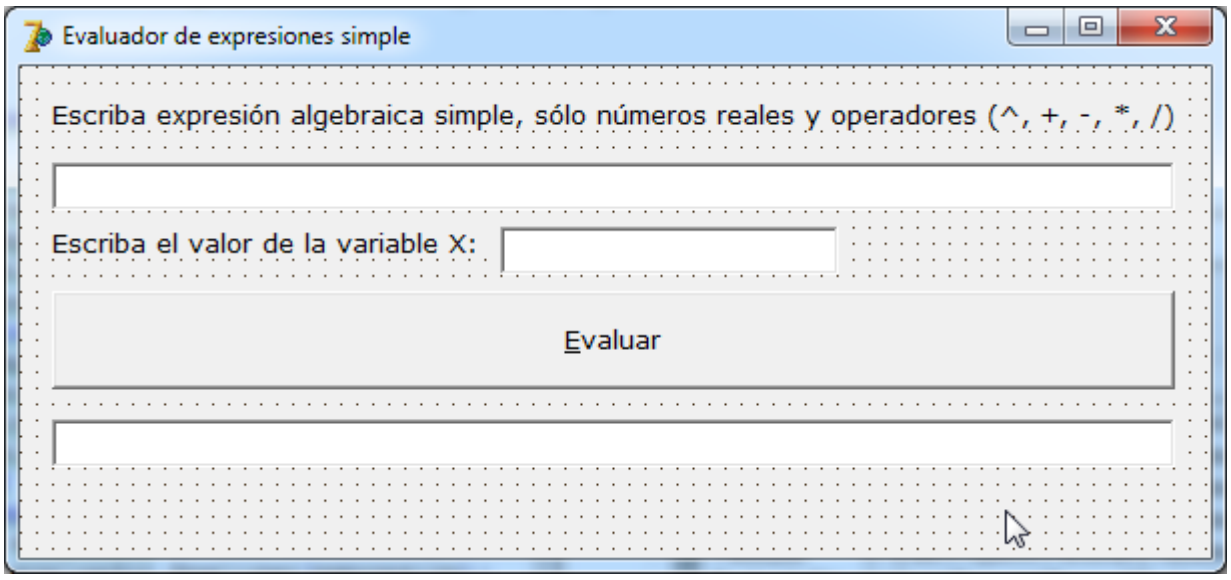
```
// Evaluación de una expresión algebraica simple:
// Sólo números reales y operadores (^, +, -, *, /)
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web:  http://darwin.50webs.com
// Licencia: LGPL
// Fecha:  Marzo de 2012
unit EvaluadorExpresionAlgebraica;

interface
uses
  //Requerido para el TobjectList
  Contnrs, SysUtils, Math, Pieza;

type
  TEvaluadorExpresionAlgebraica = class
  private
      //Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
      ESNUMERO: integer;
      ESOPERADOR: integer;
      ESVARIABLE: integer;

      //Esta constante sirve para que se reste al carácter y se obtenga el número
      //Ejemplo:  '7' - ASCIINUMERO =  7
      ASCIINUMERO: integer;

      //Esta constante sirve para que se reste al carácter y se obtenga el número
      //usado en el arreglo unidimensional que alberga los valores de las variables
      //Ejemplo:  'b' - ASCIILETRA =  1
      ASCIILETRA: integer;

      //Lista simplemente enlazada que tiene los componentes (número u operador)
      //de la expresión algebraica ya dividida.
      Piezas: TobjectList;
      objPieza: TPieza;

      //Arreglo unidimensional con las 26 variables diferentes
      valorVariable: array[0..26] of double;


      //Variable que se pone a true cuando hay un error matemático
      ERRORMATEMATICO: boolean;
  public
      Constructor Create();
      procedure Analizar(expr: string);
      function Evaluar(): double;
      function getErrorMatematico(): boolean;
      procedure DarValorVariable(variable: char; valor: double);
      procedure LiberaMemoria();
  end;

implementation

constructor TEvaluadorExpresionAlgebraica.Create();
begin
   self.ESNUMERO := 1;
   self.ESOPERADOR := 2;
   self.ESVARIABLE := 3;
   self.ASCIINUMERO := 48;
   self.ASCIILETRA := 97;
end;

//Este método se encarga de analizar la expresión y convertirla en una
//estructura que permita evaluar la expresión.
//
//Convierte una expresión algebraica en una sucesión de nodos.
//|2| |+| |8.3| |/| |5| |*| |7.4|
//
//@param expr La expresión algebraica sin espacios y en minúsculas

procedure TEvaluadorExpresionAlgebraica.Analizar(expr: string);
var
  longitud: integer;

  //Va extrayendo letra a letra de la expresión simple */
  letra: char;

  //Conversión de string a double */
  entero, fraccion, divide: integer;
  puntoentero: boolean;

  //Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero: boolean;

  cont: integer;

begin
  // Inicializa la lista de piezas
  Piezas := TObjectList.Create;
```

```
  // Tamaño de la expresión simple
  longitud := length(expr);

  // Conversión de string a double
  entero := 0;
  fraccion := 0;
  divide := 1;
  puntoentero := false;

  // Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero := false;

  for cont := 1 to longitud do
  begin
    //Trae la letra
    letra := expr[cont];

    // Si es una variable
    if letra in ['a'..'z'] then
    begin
      objPieza := TPieza.Create(ord(letra) - ASCIILETRA);
      Piezas.Add(objPieza);
    end
    // Si es un número
    else if (letra in ['0'..'9']) or (letra = '.') then
    begin
      //Ir armando el número de tipo double */
      if letra = '.' then
        puntoentero := true
      else
      begin
        if not(puntoentero) then // puntoentero = false
          entero := entero * 10 + ord(letra) - ASCIINUMERO
        else
        begin
          fraccion := fraccion * 10 + ord(letra) - ASCIINUMERO;
          divide := divide*10;
        end;
        acumulabaNumero := true;
      end
    end
    else
      begin
            // Si es un operador entonces crea el nodo del operador y el
            // nodo del número si venía acumulando un número */
            if acumulabaNumero then
            begin
              objPieza := TPieza.Create(entero + fraccion/divide);
              Piezas.Add(objPieza);
            end;

            // Agrega a la lista el operador */
            objPieza := TPieza.Create(letra);
            Piezas.Add(objPieza);

            // Inicializa de nuevo las variables de conversión de string a número */
            entero := 0;
            fraccion := 0;
            divide := 1;
            puntoentero := false;
            acumulabaNumero := false;
      end;
    end;

    // Cierra la expresión simple preguntando si el último operando es un número */
    if acumulabaNumero then
    begin
      objPieza := TPieza.Create(entero + fraccion/divide);
      Piezas.Add(objPieza);
    end;
   end;


// Ya construida la lista del tipo:
// [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
// es ir del operador con mas precedencia al de menos precedencia.
//
// Este método se llama después de haber sido analizada la expresión.
//
// @return El valor de la expresión evaluada (double)
function TEvaluadorExpresionAlgebraica.Evaluar(): double;
var
  pos, antes, sigue, totalPiezas: integer;
begin
  ERRORMATEMATICO := false;
try
  // Total de nodos en la lista creada
  totalPiezas := Piezas.Count - 1;

  for pos := 0 to totalPiezas do
  begin
    //Activa todas las piezas para que sean evaluadas */
```

```
    (Piezas.Items[pos] as TPieza).setEvaluado(false);

    // Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
    //¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
    if (Piezas.Items[pos] as Tpieza).getTipo() = ESVARIABLE then
       (Piezas.Items[pos] as Tpieza).setAcumula(valorVariable[(Piezas.Items[pos] as Tpieza).getVariable()])
    else if (Piezas.Items[pos] as Tpieza).getTipo() = ESNUMERO then
       (Piezas.Items[pos] as Tpieza).setAcumula((Piezas.Items[pos] as Tpieza).getNumero());
  end;

  // Recorre toda la lista
  for pos :=0 to  totalPiezas do
  begin
    //Si encuentra un nodo del tipo operador y es exponente */
    if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and ((Piezas.Items[pos] as Tpieza).getOperador() = '^') then
    begin
      //Busca un nodo anterior que sea número y no haya sido evaluado */
      antes := pos-1;
      while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

      //Busca un nodo siguiente que sea número y no haya sido evaluado */
      sigue := pos + 1;
      while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

      // Marca esos nodos actual y siguiente como ya evaluados */
      (Piezas.Items[pos] as Tpieza).setEvaluado(true);
      (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

       // Hace la operación de Número elevado a Número
       (Piezas.Items[antes] as Tpieza).setAcumula(Power((Piezas.Items[antes] as Tpieza).getAcumula(), (Piezas.Items[sigue] as
Tpieza).getAcumula()));
      end;
    end;

    // Recorre toda la lista
    for pos :=0 to  totalPiezas do
    begin
      // Si encuentra un nodo del tipo operador y es multiplicación o división */
      if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '*') or
((Piezas.Items[pos] as Tpieza).getOperador() = '/')) then
      begin
        //Busca un nodo anterior que sea número y no haya sido evaluado */
        antes := pos-1;
        while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

        //Busca un nodo siguiente que sea número y no haya sido evaluado */
        sigue := pos + 1;
        while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

        // Marca esos nodos actual y siguiente como ya evaluados */
        (Piezas.Items[pos] as Tpieza).setEvaluado(true);
        (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

        // Hace la operación de Número * Número, o Número / Número */
        if (Piezas.Items[pos] as Tpieza).getOperador() = '*' then
          (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() *  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
        else
        begin
          if (Piezas.Items[sigue] as Tpieza).getAcumula() <> 0 then
            (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() /  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
          else
          begin
            ERRORMATEMATICO := true;
            Evaluar := 0;
            Exit;
          end;
        end;
      end;
    end;

    // Recorre toda la lista
    for pos :=0 to  totalPiezas do
    begin
      // Si encuentra un nodo del tipo operador y es suma o resta */
      if ((Piezas.Items[pos] as Tpieza).getTipo() = ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '+') or
((Piezas.Items[pos] as Tpieza).getOperador() = '-')) then
      begin
        //Busca un nodo anterior que sea número y no haya sido evaluado */
        antes := pos-1;
        while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

        //Busca un nodo siguiente que sea número y no haya sido evaluado */
        sigue := pos + 1;
        while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

        // Marca esos nodos actual y siguiente como ya evaluados */
        (Piezas.Items[pos] as Tpieza).setEvaluado(true);
        (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

        // Hace la operación de Número + Número, o Número - Número */
```

```
      if (Piezas.Items[pos] as Tpieza).getOperador() = '+' then
        (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() +  (Piezas.Items[sigue] as
Tpieza).getAcumula() )
      else
        (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() -  (Piezas.Items[sigue] as
Tpieza).getAcumula() );
    end;
  end;

  // Resultado de la expresión
  Evaluar := (Piezas.Items[0] as Tpieza).getAcumula();

except
  on E : Exception do
  begin
    ERRORMATEMATICO := true;
    Evaluar := 0;
    exit;
  end;
end;
end;

// Guarda en un arreglo unidimensional el valor de las variables
// @param variable ¿Qué variable de la expresión recibirá el valor? a..z
// @param valor  Valor de tipo double que tendrá esa variable.
procedure TEvaluadorExpresionAlgebraica.DarValorVariable(variable: char; valor: double);
begin
         valorVariable[ord(variable) - ASCIILETRA] := valor;
end;

// Retorna true si hubo un error matemático
// @return TRUE si hubo un error matemático
function TEvaluadorExpresionAlgebraica.getErrorMatematico(): boolean;
begin
  getErrorMatematico := ERRORMATEMATICO;
end;

//Libera la memoria
procedure TEvaluadorExpresionAlgebraica.LiberaMemoria();
begin
  Piezas.Free;
end;

end.
```

Para probar ese código, podemos usar el siguiente programa



Y el código del botón sería:

```
procedure TfrmVariables.btnEvaluarClick(Sender: TObject);
var
  sintaxis: TSintaxisExpresionAlgebraica;
  evaluador: TEvaluadorExpresionAlgebraica;
  expresion: string;
  validasintaxis: integer;
  resultado, variable: double;
begin
  expresion := txtExpresion.Text;
  variable := strtofloat(txtValorX.Text);
  sintaxis :=  TSintaxisExpresionAlgebraica.Create;
  validasintaxis := sintaxis.EvaluaSintaxis(expresion);
  if validasintaxis = 0 then
  begin
    evaluador :=  TEvaluadorExpresionAlgebraica.Create;
    evaluador.Analizar(expresion);
    evaluador.DarValorVariable('x', variable);
    resultado := evaluador.Evaluar();
    if evaluador.getErrorMatematico() then
      txtResultado.Text := 'Error Matemático'
    else
      txtResultado.Text := FloattoStr(resultado);
    evaluador.LiberaMemoria();
  end
  else
    txtResultado.Text := sintaxis.MensajeSintaxis(validasintaxis);
end;
```

PARTE 4. INTERPRETACIÓN DE PARÉNTESIS.

Hasta ahora las expresiones se han evaluado teniendo en cuenta la precedencia de los operadores (primero los exponentes, luego multiplicación/división y por último la suma y resta), los paréntesis añade un cambio a esa precedencia. La forma de solucionarlo es sencilla: la expresión está a nivel cero, cada vez que se encuentra un paréntesis que abre, el nivel aumenta en uno, cuando se encuentra un paréntesis que cierra, el nivel disminuye. A continuación un ejemplo:

La expresión

```
12*(3-X)+5/(7*Y+5.4*(3.1+5.7/(4-Y)+2)-8)+5
```

Analizada por niveles quedaría así:

Nivel 0	Nivel 1	Nivel 2	Nivel 3
12*			
	3-X		
+5/			
	7*Y+5.4*		
		3.1+5.7/	
			4-Y
		+2	
	-8		
+5			

Las piezas tienen un nuevo atributo: nivel.

Durante el análisis de la expresión, cada vez que se encuentre un paréntesis que abre, se sube en uno el nivel; de allí en adelante, todo número, operador o variable estará en ese nivel, cuando se encuentre un paréntesis que cierra, se baja en uno el nivel.

Al momento de evaluar la expresión, se empieza por los niveles más altos y va bajando de nivel en nivel hasta llegar al cero.

En el ejemplo anterior se evalúa primero el Nivel 3. Estando en ese nivel, se evalúan las expresiones de ese nivel. Terminado de evaluar el Nivel 3 entonces se evalúa el Nivel 2 y así sucesivamente hasta llegar al Nivel cero.

C++

Pieza.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

/* Pieza será la unidad básica de la expresión algebraica, cada nodo tendrá o un operador
   o un número o una variable. Este nodo trabaja en una lista simplemente enlazada */
class Pieza
{
private:
    /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    int nivel;

    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
    int tipo;

    /* El número en la expresión algebraica */
    double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    char operador;

    /* La variable */
    int variable;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    bool evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    double acumula;

public:
    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     * @param nivel Nivel en que se encuentra en la expresión este número
     */
    Pieza(double, int);

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    Pieza(char, int);

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    Pieza(int, int);

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
    double getAcumula();

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    bool isEvaluado();

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    double getNumero();

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    char getOperador();

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    int getVariable();

    /**
     * Retorna que tipo de pieza es: número, operador, variable o función
```

```
     * @return Tipo de pieza
     */
    int getTipo();

    /**
     * Retorna en que nivel se encuentra la pieza con respecto a la expresión
     * @return Nivel
     */
      int getNivel();

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    void setAcumula(double);

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(bool);
};
```

Pieza.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <stdio.h>
#include <math.h>
#include "Pieza.h"

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 * @param nivel Nivel en que se encuentra en la expresión este número
 */
Pieza::Pieza(double numero, int nivel)
{
 this->tipo = 1; //Es un número
 this->numero = numero;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
Pieza::Pieza(char operador, int nivel)
{
 this->tipo = 2; //Es un operador
 this->operador = operador;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
Pieza::Pieza(int variable, int nivel)
{
 this->tipo = 3; //Es una variable
 this->variable = variable;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
double Pieza::getAcumula()
{
 return acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
```

```
 */
bool Pieza::isEvaluado()
{
 return evaluado;
}

/**
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
double Pieza::getNumero()
{
 return numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
char Pieza::getOperador()
{
 return operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
int Pieza::getVariable()
{
 return variable;
}

/**
 * Retorna que tipo de pieza es: número, operador, variable o función
 * @return Tipo de pieza
 */
int Pieza::getTipo()
{
 return tipo;
}

/**
 * Retorna en qué nivel se encuentra la pieza con respecto a la expresión
 * @return Nivel
 */
int Pieza::getNivel()
{
 return nivel;
}

/**
 * Da valor al acumulado por pieza
 * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
 */
void Pieza::setAcumula(double acumulado)
{
 this->acumula = acumulado;
}

/**
 * Marca la pieza como ya evaluada
 * @param evaluado true si la pieza ya ha sido evaluada, false si no
 */
void Pieza::setEvaluado(bool evaluado)
{
 this->evaluado = evaluado;
}
```

EvaluadorExpresionAlgebraica.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include "Pieza.h"
#include <vector>

class EvaluadorExpresionAlgebraica
{
private:
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    static const int ESNUMERO = 1;
    static const int ESOPERADOR = 2;
    static const int ESVARIABLE = 3;
```

```
    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    static const int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    static const int ASCIILETRA = 97;

    /* Lista simplemente enlazada que tiene los componentes (numero u operador
       de la expresión algebraica ya dividida */
    std::vector<Pieza> Piezas;

    /* Arreglo unidimensional con las 26 variables diferentes */
    double valorVariable[30];

    /* Almacena hasta que nivel se llega en paréntesis internos */
    int MaximoNivel;

    /* Variable que se pone a true cuando hay error matemático */
    bool ERRORMATEMATICO;

public:
    void DarValorVariable(char, double);

    /* Analiza la expresión algebraica y la vuelve una lista */
    void Analizar(char *);

    /* Evalúa la expresión que ha sido previamente analizada */
    double Evaluar(void);

    /* Retorna si hubo error matemático */
    bool getErrorMatematico();
};
```

EvaluadorExpresionAlgebraica.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

#include <math.h>
#include <stdio.h>

#include "EvaluadorExpresionAlgebraica.h"

/**
 * Convierte una expresión simple en una sucesión de nodos.
 * |2| |+| |a| |/| |5| |*| |c| |-| |1.5|
*/
void EvaluadorExpresionAlgebraica::Analizar(char *expr)
{
  /* Inicializa la lista de piezas. http://www.codeguru.com/Cpp/Cpp/cpp_mfc/stl/article.php/c4027  */
  Piezas.clear();

  /* Inicializa el nivel */
  int nivel = 0;
  MaximoNivel = 0;

  /* Conversión de string a double */
  int decimal = 0;
  int fraccion = 0;
  int divide = 1;
  bool puntodecimal = false;

  /* Tamaño de la expresión simple */
  int longitud = strlen(expr);

  /* Va extrayendo letra a letra de la expresión simple */
  char letra;

  /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
  bool acumulabaNumero = false;

  for (int cont=0; cont<longitud; cont++)
  {
    /* Trae la letra */
    letra = expr[cont];

    /* Si hay un paréntesis que abre, el nivel aumenta */
    if (letra=='(')
    {
      nivel++;
```

```
      if (nivel>MaximoNivel) MaximoNivel = nivel;
    }
    /* Si es paréntesis que cierra, el nivel disminuye */
    else if (letra==')')
      nivel--;
    /* Si es una variable */
    else if (letra >= 'a' && letra <= 'z')
    {
        Pieza objeto(letra - ASCIILETRA, nivel);
        Piezas.push_back(objeto);
    }
    /* Si es un número */
    else if( (letra >= '0' && letra <= '9') || letra == '.')
    {
      /* Ir armando el número de tipo double */
      if (letra == '.')
        puntodecimal = true;
      else
        if(!puntodecimal)  /* puntodecimal == false */
          decimal = decimal * 10 + letra-ASCIINUMERO;
        else
        {
          fraccion = fraccion * 10 + letra-ASCIINUMERO;
          divide *=10;
        }
      acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
         nodo del número si venía acumulando un número */
      if (acumulabaNumero)
      {
        /* El nodo del número */
        Pieza objeto(decimal + (double) fraccion/divide, nivel);

        /* Agrega a la lista */
        Piezas.push_back(objeto);
      }

      /* El nodo operador */
      Pieza objeto(letra, nivel);

      /* Agrega a la lista */
      Piezas.push_back(objeto);

      /* Inicializa de nuevo las variables de conversión de string a número */
      decimal = 0;
      fraccion = 0;
      divide = 1;
      puntodecimal = false;
      acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    Pieza objeto(decimal + (double) fraccion/divide, nivel);
    Piezas.push_back(objeto);
  }

}

/* Ya construida la lista del tipo:
   [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   es ir del operador con mas precedencia al de menos precedencia */
double EvaluadorExpresionAlgebraica::Evaluar()
{
  int pos=0, antes=0, sigue=0;
  ERRORMATEMATICO = false;

  /* Total de nodos en la lista creada */
  int totalPiezas = Piezas.size();

  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Activa todas las piezas para que sean evaluadas */
    Piezas[pos].setEvaluado(false);

    /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
      ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
    if (Piezas[pos].getTipo() == ESVARIABLE)
      Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
    else if (Piezas[pos].getTipo() == ESNUMERO)
      Piezas[pos].setAcumula(Piezas[pos].getNumero());
  }

  /* Va del nivel mas profundo al mas superficial */
  for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
  {
    /* Recorre toda la lista */
```

```
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es exponente */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número elevado a Número */
        Piezas[antes].setAcumula(pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es multiplicación o división */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' ||
Piezas[pos].getOperador() == '/'))
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número * Número, o Número / Número */
        if (Piezas[pos].getOperador() == '*')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
        else
        {
          if (Piezas[sigue].getAcumula() != 0)
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
        }
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es suma o resta */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' ||
Piezas[pos].getOperador() == '-'))
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número + Número, o Número - Número */
        if (Piezas[pos].getOperador() == '+')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
        else
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
      }
    }
  }

  /* Resultado de la expresión */
  return Piezas[0].getAcumula();
}

/* Guarda en un arreglo unidimensional el valor de las variables */
void EvaluadorExpresionAlgebraica::DarValorVariable(char variable, double valor)
{
   valorVariable[variable - ASCIILETRA] = valor;
}

/* Retorna true si hubo un error matemático */
bool EvaluadorExpresionAlgebraica::getErrorMatematico()
{
```

```
  return ERRORMATEMATICO;
}
```

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
    char expresion[500];
    EvaluadorExpresionAlgebraica objEvaluador;
    double valor;

    strcpy(expresion, "(x)+(3.905/8.36/x/1.953-9.556+x)-x");
    objEvaluador.Analizar(expresion);
    objEvaluador.DarValorVariable('x', 13.1729);
    valor = objEvaluador.Evaluar();
    printf("%f\n", valor);
    getchar();
    return 0;
}
```

C#

Pieza.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    public class Pieza
    {
        /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
        private int nivel;

        /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
        private int tipo;

        /* El número en la expresión algebraica */
        private double numero;

        /* El operador (+, -, *, /, ^) en la expresión algebraica */
        private char operador;

        /* La variable */
        private int variable;

        /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
           cálculos que determinan el valor de la expresión */
        private bool evaluado;

        /* El resultado parcial de la evaluación de la expresión */
        private double acumula;

        /**
         * Constructor en caso que la pieza contenga un número real
         * @param numero Número de tipo double
         * @param nivel Nivel en que se encuentra en la expresión este número
         */
        public Pieza(double numero, int nivel)
        {
            this.tipo = 1; //Es un número
            this.numero = numero;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = numero;
        }

        /**
         * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
         * @param operador Que puede ser +, -, *, /, ^
         * @param nivel Nivel en que se encuentra en la expresión este operador
         */
        public Pieza(char operador, int nivel)
        {
            this.tipo = 2; //Es un operador
            this.operador = operador;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = 0;
        }

        /**
         * Constructor en caso que el nodo contenga una variable
         * @param variable Puede ir de 0 a 25 que representa de a..z
         * @param nivel Nivel en que se encuentra en la expresión este operador
         */
        public Pieza(int variable, int nivel)
        {
            this.tipo = 3; //Es una variable
            this.variable = variable;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = 0;
        }

        /** Retorna el acumulado que tiene esta pieza
         * @return Acumulado de la pieza
         */
```

```
        public double getAcumula()
        {
            return acumula;
        }

        /** Retorna si la pieza ya fue evaluada
         * @return True si la pieza ya fue evaluada
         */
        public bool isEvaluado()
        {
            return evaluado;
        }

        /**
         * Retorna el número de tipo double que tiene la pieza
         * @return El valor del número tipo double
         */
        public double getNumero()
        {
            return numero;
        }

        /**
         * Retorna el operador (+, -, *, /, ^) que tiene la pieza
         * @return El operador en char
         */
        public char getOperador()
        {
            return operador;
        }

        /**
         * Retorna la variable que tiene la pieza
         * @return La variable
         */
        public int getVariable()
        {
            return variable;
        }

        /**
         * Retorna que tipo de pieza es: número, operador, variable o función
         * @return Tipo de pieza
         */
        public int getTipo()
        {
            return tipo;
        }

        /**
         * Retorna en que nivel se encuentra la pieza con respecto a la expresión
         * @return Nivel
         */
        public int getNivel()
        {
            return nivel;
        }

        /**
         * Da valor al acumulado por pieza
         * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
         */
        public void setAcumula(double acumulado)
        {
            this.acumula = acumulado;
        }

         /**
         * Marca la pieza como ya evaluada
         * @param evaluado true si la pieza ya ha sido evaluada, false si no
         */
        public void setEvaluado(bool evaluado)
        {
            this.evaluado = evaluado;
        }
    }
}
```

EvaluadorExpresionAlgebraica.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

namespace EvaluadorCSharp
{
    class EvaluadorExpresionAlgebraica
    {
        /**
         * Constantes para determinar qué tipo es cada pieza en que se divide la expresión algebraica
         */
        private static int ESNUMERO = 1;
        private static int ESOPERADOR = 2;
        private static int ESVARIABLE = 3;

        /**
         * Esta constante sirve para que se reste al carácter y se obtenga el número
         * Ejemplo:  '7' - ASCIINUMERO =  7
         */
        private static int ASCIINUMERO = 48;

        /**
         * Esta constante sirve para que se reste al carácter y se obtenga el número
         * usado en el arreglo unidimensional que alberga los valores de las variables
         * Ejemplo:  'b' - ASCIILETRA =  1
         */
        private static int ASCIILETRA = 97;

        /**
         * Lista simplemente enlazada que tiene los componentes (número, operador o función)
         * de la expresión algebraica ya dividida.
         */
        private List<Pieza> Piezas = new List<Pieza>();

        /**
         * Arreglo unidimensional con las 26 variables diferentes
         */
        private double[] valorVariable = new double[26];

        /**
         * Almacena hasta que nivel se llega en paréntesis internos
         */
        private int MaximoNivel;

        /**
         * Variable que se pone a true cuando hay un error matemático
         */
        private bool ERRORMATEMATICO;

        /**
         * Este método se encarga de analizar la expresión y convertirla en una
         * estructura que permita evaluar la expresión.
         *
         * Convierte una expresión algebraica en una sucesión de nodos.
         * |2| |+| |a| |/| |5| |*| |cos| |y|
         *
         * @param expr La expresión algebraica sin espacios y en minúsculas
        */
        public void Analizar(String expr)
        {
            /* Inicializa la lista de piezas  */
            Piezas.Clear();

            /* Inicializa el nivel */
            int nivel = 0;
            MaximoNivel = 0;

            /* Tamaño de la expresión simple */
            int longitud = expr.Length;

            /* Va extrayendo letra a letra de la expresión simple */
            char letra;

            /* Conversión de string a double */
            int entero = 0, fraccion = 0, divide = 1;
            bool puntoentero = false;

            /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
            bool acumulabaNumero = false;

            for (int cont=0; cont<longitud; cont++)
            {
                /* Trae la letra */
                letra = expr[cont];

                /* Si hay un paréntesis que abre, el nivel aumenta */
                if (letra=='(')
                {
                    nivel++;
                    if (nivel>MaximoNivel) MaximoNivel = nivel;
                }
                /* Si es paréntesis que cierra, el nivel disminuye */
                else if (letra==')')
                    nivel--;
```

```
            /* Si es una variable */
            else if (letra >= 'a' && letra <= 'z')
            {
                    Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
                    Piezas.Add(objeto);
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntoentero = true;
                else
                    if(!puntoentero)  /* puntoentero == false */
                        entero = entero * 10 + letra-ASCIINUMERO;
                    else
                    {
                        fraccion = fraccion * 10 + letra-ASCIINUMERO;
                        divide *=10;
                    }
                    acumulabaNumero = true;
            }
            else
            {
                /* Si es un operador entonces crea el nodo del operador y el
                    nodo del número si venía acumulando un número */
                if (acumulabaNumero)
                {
                    /* El nodo del número */
                    Pieza objeto = new Pieza(entero + (double) fraccion/divide, nivel);

                    /* Agrega a la lista */
                    Piezas.Add(objeto);
                }

                /* El nodo operador */
                Pieza objeto2 = new Pieza(letra, nivel);

                /* Agrega a la lista */
                Piezas.Add(objeto2);

                /* Inicializa de nuevo las variables de conversión de string a número */
                entero = fraccion = 0;
                divide = 1;
                puntoentero = acumulabaNumero = false;
            }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
        {
            Pieza objeto = new Pieza(entero + (double) fraccion/divide, nivel);
            Piezas.Add(objeto);
        }
    }

    /**
     * Ya construida la lista del tipo:
     * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
     * es ir del operador con más precedencia al de menos precedencia.
     *
     * Este método se llama después de haber sido analizada la expresión.
     *
     * En el caso que sólo cambia el valor de una variable, no es necesario
     * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
     * la misma expresión.
     *
     * @return El valor de la expresión evaluada (double)
     */
    public double Evaluar()
    {
        int pos=0, antes=0, sigue=0;
        ERRORMATEMATICO = false;

        /* Total de nodos en la lista creada */
        int totalPiezas = Piezas.Count;

        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Activa todas las piezas para que sean evaluadas */
            Piezas[pos].setEvaluado(false);

            /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
               ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
            if (Piezas[pos].getTipo() == ESVARIABLE)
                Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
            else if (Piezas[pos].getTipo() == ESNUMERO)
                Piezas[pos].setAcumula(Piezas[pos].getNumero());
        }

        /* Va del nivel más profundo al más superficial */
        for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
```

```
            {
                /* Recorre toda la lista */
                for (pos=0; pos<totalPiezas; pos++)
                {
                    /* Si encuentra un nodo del tipo operador y es exponente */
                    if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR &&
Piezas[pos].getOperador() == '^')
                    {
                        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

                        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

                        /* Marca esos nodos actual y siguiente como ya evaluados */
                        Piezas[pos].setEvaluado(true);
                        Piezas[sigue].setEvaluado(true);

                        /* Hace la operación de Número elevado a Número */
                        Piezas[antes].setAcumula(Math.Pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
                    }
                }

                /* Recorre toda la lista */
                for (pos=0; pos<totalPiezas; pos++)
                {
                    /* Si encuentra un nodo del tipo operador y es multiplicación o división */
                    if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR &&
(Piezas[pos].getOperador() == '*' || Piezas[pos].getOperador() == '/'))
                    {
                        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

                        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

                        /* Marca esos nodos actual y siguiente como ya evaluados */
                        Piezas[pos].setEvaluado(true);
                        Piezas[sigue].setEvaluado(true);

                        /* Hace la operación de Número * Número, o Número / Número */
                        if (Piezas[pos].getOperador() == '*')
                            Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
                        else
                        {
                            if (Piezas[sigue].getAcumula() != 0)
                                Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
                            else
                            {
                                ERRORMATEMATICO = true;
                                return 0;
                            }
                        }

                    }
                }

                /* Recorre toda la lista */
                for (pos=0; pos<totalPiezas; pos++)
                {
                    /* Si encuentra un nodo del tipo operador y es suma o resta */
                    if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR &&
(Piezas[pos].getOperador() == '+' || Piezas[pos].getOperador() == '-'))
                    {
                        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

                        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

                        /* Marca esos nodos actual y siguiente como ya evaluados */
                        Piezas[pos].setEvaluado(true);
                        Piezas[sigue].setEvaluado(true);

                        /* Hace la operación de Número + Número, o Número - Número */
                        if (Piezas[pos].getOperador() == '+')
                            Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
                        else
                            Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
                    }
                }
            }

            /* Retorna el resultado */
            return Piezas[0].getAcumula();
        }

        /**
         * Guarda en un arreglo unidimensional el valor de las variables
         * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
         * @param valor  Valor de tipo double que tendrá esa variable.
         */
```

```
        public void DarValorVariable(char variable, double valor)
        {
             valorVariable[variable - ASCIILETRA] = valor;
        }

        /**
         * Retorna true si hubo un error matemático
         * @return TRUE si hubo un error matemático
         */
        public bool getErrorMatematico()
        {
                return ERRORMATEMATICO;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
            string expresion;
            double valor;

            expresion = "4.448/(x)+(x/4.732/5.181*x)-x*(x*2.771)-(4.459-4.870)";
            objEvaluador.Analizar(expresion);
            objEvaluador.DarValorVariable('x', 13.1729);
            valor = objEvaluador.Evaluar();
            Console.WriteLine(valor);
            Console.ReadKey();
        }
    }
}
```

Visual Basic .NET

Pieza.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web:  http://darwin.50webs.com
'Licencia: LGPL
'Fecha:  Enero de 2012

Public Class Pieza
    'Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión
    Private nivel As Integer

    '1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función
    Private tipo As Integer

    'El número en la expresión algebraica
    Private numero As Double

    'El operador (+, -, *, /, ^) en la expresión algebraica
    Private operador As Char

    'La variable
    Private variable As Integer

    'Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    'cálculos que determinan el valor de la expresión
    Private evaluado As Boolean

    'El resultado parcial de la evaluación de la expresión
    Private acumula As Double

    'Constructor en caso que la pieza contenga un número real
    '@param numero Número de tipo dim  as double
    '@param nivel Nivel en que se encuentra en la expresión este número
    Public Sub New(ByVal numero As Double, ByVal nivel As Integer)
        Me.tipo = 1 'Es un número
        Me.numero = numero
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = numero
    End Sub

    'Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
    '@param operador Que puede ser +, -, *, /, ^
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal operador As Char, ByVal nivel As Integer)

        Me.tipo = 2 'Es un operador
        Me.operador = operador
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = 0
    End Sub


    'Constructor en caso que el nodo contenga una variable
    '@param variable Puede ir de 0 a 25 que representa de a..z
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal variable As Integer, ByVal nivel As Integer)
        Me.tipo = 3 'Es una variable
        Me.variable = variable
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = 0
    End Sub


    'Retorna el acumulado que tiene esta pieza
    '@return Acumulado de la pieza
    Public Function getAcumula() As Double
        Return acumula
    End Function

    'Retorna si la pieza ya fue evaluada
    '@return True si la pieza ya fue evaluada
    Public Function isEvaluado() As Boolean
        Return evaluado
    End Function


    'Retorna el número de tipo dim  as double que tiene la pieza
    '@return El valor del número tipo dim  as double
    Public Function getNumero() As Double
        Return numero
    End Function


    'Retorna el operador (+, -, *, /, ^) que tiene la pieza
```

```
    '@return El operador en dim  as char
    Public Function getOperador() As Char
        Return operador
    End Function


    'Retorna la variable que tiene la pieza
    '@return La variable
    Public Function getVariable() As Integer
        Return variable
    End Function


    'Retorna que tipo de pieza es: número, operador, variable o función
    '@return Tipo de pieza
    Public Function getTipo() As Integer
        Return tipo
    End Function


    'Retorna en qué nivel se encuentra la pieza con respecto a la expresión
    '@return Nivel
    Public Function getNivel() As Integer
        Return nivel
    End Function


    'Da valor al acumulado por pieza
    '@param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
    Public Sub setAcumula(ByVal acumulado As Double)
        Me.acumula = acumulado
    End Sub

    'Marca la pieza como ya evaluada
    '@param evaluado true si la pieza ya ha sido evaluada, false si no
    Public Sub setEvaluado(ByVal evaluado As Boolean)
        Me.evaluado = evaluado
    End Sub

End Class
```

EvaluadorExpresionAlgebraica.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web: http://darwin.50webs.com
'Licencia: LGPL
'Fecha: Enero de 2012

Public Class EvaluadorExpresionAlgebraica
  'Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
  Private Shared ESNUMERO As Integer = 1
  Private Shared ESOPERADOR As Integer = 2
  Private Shared ESVARIABLE As Integer = 3

  'Esta constante sirve para que se reste al caracter y se obtenga el número
  'Ejemplo: '7' - ASCIINUMERO = 7
  Private Shared ASCIINUMERO As Integer = 48

  'Esta constante sirve para que se reste al caracter y se obtenga el número
  'usado en el arreglo unidimensional que alberga los valores de las variables
  'Ejemplo: 'b' - ASCIILETRA = 1
  Private Shared ASCIILETRA As Integer = 97

  'Lista simplemente enlazada que tiene los componentes (número, operador o función)
  'de la expresión algebraica ya dividida.
  Private Piezas As New List(Of Pieza)()

  'Arreglo unidimensional con las 26 variables diferentes
  Private valorVariable As Double() = New Double(25) {}

  'Almacena hasta que nivel se llega en paréntesis internos
  Private MaximoNivel As Integer

  'Variable que se pone a true cuando hay un error matemático
  Private ERRORMATEMATICO As Boolean

  'Este método se encarga de analizar la expresión y convertirla en una
  'estructura que permita evaluar la expresión.
  '     *
  'Convierte una expresión algebraica en una sucesión de nodos.
  '|2| |+| |a| |/| |5| |*| |cos| |y|
  '|2| |+| |a| |/| |5| |*| |cos| |y|
  '
  '@param expr La expresión algebraica sin espacios y en minúsculas
  Public Sub Analizar(ByVal expr As [String])
    ' Inicializa la lista de piezas
    Piezas.Clear()

    ' Inicializa el nivel
    Dim nivel As Integer = 0
    MaximoNivel = 0
```

```
    ' Tamaño de la expresión simple
    Dim longitud As Integer = expr.Length

    ' Va extrayendo letra a letra de la expresión simple
    Dim letra As Char

    ' Conversión de string a double
    Dim entero As Integer = 0, fraccion As Integer = 0, divide As Integer = 1
    Dim puntoentero As Boolean = False

    ' Si llevaba acumulando un valor numérico esta variable se vuelve true
    Dim acumulabaNumero As Boolean = False

    For cont As Integer = 0 To longitud - 1
      ' Trae la letra
      letra = expr(cont)

      ' Si hay un paréntesis que abre, el nivel aumenta
      If letra = "(" Then
        nivel += 1
        If nivel > MaximoNivel Then
          MaximoNivel = nivel
        End If
        ' Si es paréntesis que cierra, el nivel disminuye
      ElseIf letra = ")" Then
        nivel -= 1
        ' Si es una variable o una función
      ElseIf letra >= "a" And letra <= "z" Then
        ' Es una variable, no una función
        Dim objeto As New Pieza(Asc(letra) - ASCIILETRA, nivel)
        Piezas.Add(objeto)
        ' Si es un número
      ElseIf (letra >= "0" And letra <= "9") Or letra = "." Then
        ' Ir armando el número de tipo double
        If letra = "." Then
          puntoentero = True
        ElseIf Not puntoentero Then
          ' puntoentero == false
          entero = entero * 10 + Asc(letra) - ASCIINUMERO
        Else
          fraccion = fraccion * 10 + Asc(letra) - ASCIINUMERO
          divide *= 10
        End If
        acumulabaNumero = True
      Else
        ' Si es un operador entonces crea el nodo del operador y el
        ' nodo del número si venía acumulando un número

        If acumulabaNumero Then
          ' El nodo del número
          Dim objeto As New Pieza(entero + CDbl(fraccion) / divide, nivel)

          ' Agrega a la lista
          Piezas.Add(objeto)
        End If

        ' El nodo operador
        Dim objeto2 As New Pieza(letra, nivel)

        ' Agrega a la lista
        Piezas.Add(objeto2)

        ' Inicializa de nuevo las variables de conversión de string a número
        entero = 0
        fraccion = 0
        divide = 1
        puntoentero = False
        acumulabaNumero = False
      End If
    Next

    ' Cierra la expresión simple preguntando si el último operando es un número
    If acumulabaNumero Then
      Dim objeto As New Pieza(entero + CDbl(fraccion) / divide, nivel)
      Piezas.Add(objeto)
    End If
  End Sub

  'Ya construida la lista del tipo:
  '[nodo número] [nodo operador] [nodo número] [nodo operador] .....
  'es ir del operador con más precedencia al de menos precedencia.
  '
  'Este método se llama después de haber sido analizada la expresión.
  '
  'En el caso que sólo cambia el valor de una variable, no es necesario
  'analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
  'la misma expresión.
  '
  '@return El valor de la expresión evaluada (double)
  Public Function Evaluar() As Double
    Dim pos As Integer = 0, antes As Integer = 0, sigue As Integer = 0
```

```
    ERRORMATEMATICO = False

    ' Total de nodos en la lista creada
    Dim totalPiezas As Integer = Piezas.Count

    For pos = 0 To totalPiezas - 1
      ' Activa todas las piezas para que sean evaluadas
      Piezas(pos).setEvaluado(False)

      ' Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
      ' ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables.
      If Piezas(pos).getTipo() = ESVARIABLE Then
        Piezas(pos).setAcumula(valorVariable(Piezas(pos).getVariable()))
      ElseIf Piezas(pos).getTipo() = ESNUMERO Then
        Piezas(pos).setAcumula(Piezas(pos).getNumero())
      End If
    Next

    ' Va del nivel mas profundo al mas superficial
    For evaluaNivel As Integer = MaximoNivel To 0 Step -1

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es exponente
        If Piezas(pos).getNivel() = evaluaNivel And Piezas(pos).getTipo() = ESOPERADOR And Piezas(pos).getOperador() = "^" Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While

          ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
          sigue = pos + 1
          While Piezas(sigue).isEvaluado()
            sigue += 1
          End While

          ' Marca esos nodos actual y siguiente como ya evaluados
          Piezas(pos).setEvaluado(True)
          Piezas(sigue).setEvaluado(True)

          ' Hace la operación de Número elevado a Número
          Piezas(antes).setAcumula(Math.Pow(Piezas(antes).getAcumula(), Piezas(sigue).getAcumula()))
        End If
      Next

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es multiplicación o división
        If Piezas(pos).getNivel() = evaluaNivel And Piezas(pos).getTipo() = ESOPERADOR And (Piezas(pos).getOperador() = "*" Or
Piezas(pos).getOperador() = "/") Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While

          ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
          sigue = pos + 1
          While Piezas(sigue).isEvaluado()
            sigue += 1
          End While

          ' Marca esos nodos actual y siguiente como ya evaluados
          Piezas(pos).setEvaluado(True)
          Piezas(sigue).setEvaluado(True)

          ' Hace la operación de Número * Número, o Número / Número
          If Piezas(pos).getOperador() = "*" Then
            Piezas(antes).setAcumula(Piezas(antes).getAcumula() * Piezas(sigue).getAcumula())
          Else
            If Piezas(sigue).getAcumula() <> 0 Then
              Piezas(antes).setAcumula(Piezas(antes).getAcumula() / Piezas(sigue).getAcumula())
            Else
              ERRORMATEMATICO = True
              Return 0
            End If

          End If
        End If
      Next

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es suma o resta
        If Piezas(pos).getNivel() = evaluaNivel And Piezas(pos).getTipo() = ESOPERADOR And (Piezas(pos).getOperador() = "+" Or
Piezas(pos).getOperador() = "-") Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While
```

```
        ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
        sigue = pos + 1
        While Piezas(sigue).isEvaluado()
          sigue += 1
        End While

        ' Marca esos nodos actual y siguiente como ya evaluados
        Piezas(pos).setEvaluado(True)
        Piezas(sigue).setEvaluado(True)

        ' Hace la operación de Número + Número, o Número - Número
        If Piezas(pos).getOperador() = "+" Then
          Piezas(antes).setAcumula(Piezas(antes).getAcumula() + Piezas(sigue).getAcumula())
        Else
          Piezas(antes).setAcumula(Piezas(antes).getAcumula() - Piezas(sigue).getAcumula())
        End If
      End If
    Next
  Next

  ' Resultado de la evaluación
  Return Piezas(0).getAcumula()
End Function

'Guarda en un arreglo unidimensional el valor de las variables
'@param variable ¿Qué variable de la expresión recibirá el valor? a..z
'@param valor Valor de tipo double que tendrá esa variable.
Public Sub DarValorVariable(ByVal variable As Char, ByVal valor As Double)
  valorVariable(Asc(variable) - ASCIILETRA) = valor
End Sub

'Retorna true si hubo un error matemático
'@return TRUE si hubo un error matemático
Public Function getErrorMatematico() As Boolean
  Return ERRORMATEMATICO
End Function

End Class
```

```
Module Module1

    Sub Main()
        Dim objEvaluador As New EvaluadorExpresionAlgebraica
        Dim valor, valorreal As Double
        Dim expresion As String

        expresion = "x/((7.43*6.67/(6.108+(x+x*x)))"
        objEvaluador.Analizar(expresion)
        objEvaluador.DarValorVariable("x", 13.1729)
        valor = objEvaluador.Evaluar()
        Console.WriteLine(valor)
        Console.ReadKey()
    End Sub

End Module
```

Java

Pieza.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
public class Pieza
{
    /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    private int nivel;

    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
    private int tipo;

    /* El número en la expresión algebraica */
    private double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    private char operador;

    /* La variable */
    private int variable;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    private boolean evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    private double acumula;

    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     * @param nivel Nivel en que se encuentra en la expresión este número
     */
    public Pieza(double numero, int nivel)
    {
     this.tipo = 1; //Es un número
     this.numero = numero;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = numero;
    }

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(char operador, int nivel)
    {
     this.tipo = 2; //Es un operador
     this.operador = operador;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = 0;
    }

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(int variable, int nivel)
    {
     this.tipo = 3; //Es una variable
     this.variable = variable;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = 0;
    }

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
    public double getAcumula()
    {
     return acumula;
    }

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    public boolean isEvaluado()
    {
```

```
     return evaluado;
    }

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    public double getNumero()
    {
     return numero;
    }

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    public char getOperador()
    {
     return operador;
    }

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    public int getVariable()
    {
     return variable;
    }

    /**
     * Retorna que tipo de pieza es: número, operador o variable
     * @return Tipo de pieza
     */
    public int getTipo()
    {
     return tipo;
    }

    /**
     * Retorna en que nivel se encuentra la pieza con respecto a la expresión
     * @return Nivel
     */
    public int getNivel()
    {
            return nivel;
    }

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    public void setAcumula(double acumulado)
    {
     this.acumula = acumulado;
    }

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(boolean evaluado)
    {
     this.evaluado = evaluado;
    }
}
```

EvaluadorExpresionAlgebraica.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

import java.util.ArrayList;

public class EvaluadorExpresionAlgebraica
{
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    private static final int ESNUMERO = 1;
    private static final int ESOPERADOR = 2;
    private static final int ESVARIABLE = 3;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
```

```
     */
    private static final int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    private static final int ASCIILETRA = 97;

    /**
     * Lista simplemente enlazada que tiene los componentes (número u operador)
     * de la expresión algebraica ya dividida.
     */
    private ArrayList<Pieza> Piezas = new ArrayList<Pieza>();

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    private double valorVariable[] = new double[26];

    /**
     * Almacena hasta que nivel se llega en paréntesis internos
     */
    private int MaximoNivel;

    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    private boolean ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    public void Analizar(String expr)
    {
        /* Inicializa la lista de piezas  */
        Piezas.clear();

        /* Inicializa el nivel */
        int nivel = 0;
        MaximoNivel = 0;

        /* Tamaño de la expresión simple */
        int longitud = expr.length();

        /* Va extrayendo letra a letra de la expresión simple */
        char letra;

        /* Conversión de string a double */
        int decimal = 0, fraccion = 0, divide = 1;
        boolean puntodecimal = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        boolean acumulabaNumero = false;

        for (int cont=0; cont<longitud; cont++)
        {
            /* Trae la letra */
            letra = expr.charAt(cont);

            /* Si hay un paréntesis que abre, el nivel aumenta */
            if (letra=='(')
            {
                nivel++;
                if (nivel>MaximoNivel) MaximoNivel = nivel;
            }
            /* Si es paréntesis que cierra, el nivel disminuye */
            else if (letra==')')
                nivel--;
            /* Si es una variable */
            else if (letra >= 'a' && letra <= 'z')
            {
                Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
                Piezas.add(objeto);
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntodecimal = true;
                else
                    if(!puntodecimal)  /* puntodecimal == false */
                        decimal = decimal * 10 + letra-ASCIINUMERO;
                    else
```

```
                {
                    fraccion = fraccion * 10 + letra-ASCIINUMERO;
                    divide *=10;
                }
                acumulabaNumero = true;
            }
            else
            {
                /* Si es un operador entonces crea el nodo del operador y el
                   nodo del número si venía acumulando un número */
                if (acumulabaNumero)
                {
                    /* El nodo del número */
                    Pieza objeto = new Pieza(decimal + (double) fraccion/divide, nivel);

                    /* Agrega a la lista */
                    Piezas.add(objeto);
                }

                /* El nodo operador */
                Pieza objeto = new Pieza(letra, nivel);

                /* Agrega a la lista */
                Piezas.add(objeto);

                /* Inicializa de nuevo las variables de conversión de string a número */
                decimal = fraccion = 0;
                divide = 1;
                puntodecimal = acumulabaNumero = false;
            }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
        {
            Pieza objeto = new Pieza(decimal + (double) fraccion/divide, nivel);
            Piezas.add(objeto);
        }
    }

    /**
     * Ya construida la lista del tipo:
     * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
     * es ir del operador con mas precedencia al de menos precedencia.
     *
     * Este método se llama después de haber sido analizada la expresión.
     *
     * En el caso que sólo cambia el valor de una variable, no es necesario
     * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
     * la misma expresión.
     *
     * @return El valor de la expresión evaluada (double)
     */
    public double Evaluar()
    {
        int pos=0, antes=0, sigue=0;
        ERRORMATEMATICO = false;

        /* Total de nodos en la lista creada */
        int totalPiezas = Piezas.size();

        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Activa todas las piezas para que sean evaluadas */
            Piezas.get(pos).setEvaluado(false);

            /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
               ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
            if (Piezas.get(pos).getTipo() == ESVARIABLE)
                Piezas.get(pos).setAcumula(valorVariable[Piezas.get(pos).getVariable()]);
            else if (Piezas.get(pos).getTipo() == ESNUMERO)
                Piezas.get(pos).setAcumula(Piezas.get(pos).getNumero());
        }

        /* Va del nivel mas profundo al mas superficial */
        for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
        {
            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es exponente */
                if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
Piezas.get(pos).getOperador() == '^')
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    Piezas.get(pos).setEvaluado(true);
```

```
                    Piezas.get(sigue).setEvaluado(true);

                    /* Hace la operación de Número elevado a Número */
                    Piezas.get(antes).setAcumula(Math.pow(Piezas.get(antes).getAcumula(), Piezas.get(sigue).getAcumula()));
                }
            }

            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es multiplicación o división */
                if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
(Piezas.get(pos).getOperador() == '*' || Piezas.get(pos).getOperador() == '/'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    Piezas.get(pos).setEvaluado(true);
                    Piezas.get(sigue).setEvaluado(true);

                    /* Hace la operación de Número * Número, o Número / Número */
                    if (Piezas.get(pos).getOperador() == '*')
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() *  Piezas.get(sigue).getAcumula() );
                    else
                    {
                        if (Piezas.get(sigue).getAcumula() != 0)
                            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() /  Piezas.get(sigue).getAcumula() );
                        else
                        {
                            ERRORMATEMATICO = true;
                            return 0;
                        }
                    }

                }
            }

            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es suma o resta */
                if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
(Piezas.get(pos).getOperador() == '+' || Piezas.get(pos).getOperador() == '-'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    Piezas.get(pos).setEvaluado(true);
                    Piezas.get(sigue).setEvaluado(true);

                    /* Hace la operación de Número + Número, o Número - Número */
                    if (Piezas.get(pos).getOperador() == '+')
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() +  Piezas.get(sigue).getAcumula() );
                    else
                        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() -  Piezas.get(sigue).getAcumula() );
                }
            }
        }

        /* Resultado de la expresión */
        return Piezas.get(0).getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    public void DarValorVariable(char variable, double valor)
    {
         valorVariable[variable - ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    public boolean getErrorMatematico()
    {
            return ERRORMATEMATICO;
    }
}
```

```
public class Inicio
{
    public static void main(String args[])
    {
        EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
        String expresion;
        double valor;

        expresion = "(5.940-7.378)/x+9.756*5.43+9.125/x";
        objEvaluador.Analizar(expresion);
        objEvaluador.DarValorVariable('x', 13.1729);
        valor = objEvaluador.Evaluar();
        System.out.println(valor);
    }
}
```

PHP

Pieza.php

```php
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
class Pieza
{
  /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
  var $nivel;

  /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
  var $tipo;

  /* El número en la expresión algebraica */
  var $numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  var $operador;

  /* La variable */
  var $variable;

/* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
   cálculos que determinan el valor de la expresión */
var $evaluado;

/* El resultado parcial de la evaluación de la expresión */
var $acumula;

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 */
public function ConstructorNumero($numero, $nivel)
{
 $this->tipo = 1; //Es un número
 $this->numero = $numero;
 $this->nivel = $nivel;

 $this->evaluado = false;
 $this->acumula = $numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 */
public function ConstructorOperador($operador, $nivel)
{
  $this->tipo = 2; //Es un operador
  $this->operador = $operador;
  $this->nivel = $nivel;

  $this->evaluado = false;
  $this->acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
public function ConstructorVariable($variable, $nivel)
{
  $this->tipo = 3; //Es una variable
  $this->variable = $variable;
  $this->nivel = $nivel;

  $this->evaluado = false;
  $this->acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
public function getAcumula()
{
  return $this->acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
function isEvaluado()
```

```
{
  return $this->evaluado;
}

/**
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
public function getNumero()
{
  return $this->numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
public function getOperador()
{
  return $this->operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
public function getVariable()
{
 return $this->variable;
}

/**
 * Retorna que tipo de pieza es: número u operador
 * @return Tipo de pieza
 */
public function getTipo()
{
  return $this->tipo;
}

/**
 * Retorna en que nivel se encuentra la pieza con respecto a la expresión
 * @return Nivel
 */
public function getNivel()
{
   return $this->nivel;
}

/**
 * Retorna el código de la función (abs, sen, cos, tan) que tiene la piez
 * @return Código de la función
 */
public function getFuncion()
{
   return $this->funcion;
}

/**
 * Da valor al acumulado por pieza
 * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
 */
public function setAcumula($acumulado)
{
  $this->acumula = $acumulado;
}

/**
 * Marca la pieza como ya evaluada
 * @param evaluado true si la pieza ya ha sido evaluada, false si no
 */
public function setEvaluado($evaluado)
{
  $this->evaluado = $evaluado;
}
}
?>
```

EvaluadorExpresionAlgebraica.php

```
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
include("Pieza.php");

class EvaluadorExpresionAlgebraica
```

```
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  var $ESNUMERO = 1;
  var $ESOPERADOR = 2;
  var $ESVARIABLE = 3;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * Ejemplo:  '7' - ASCIINUMERO =  7
   */
  var $ASCIINUMERO = 48;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * usado en el arreglo unidimensional que alberga los valores de las variables
   * Ejemplo:  'b' - ASCIILETRA =  1
   */
  var $ASCIILETRA = 97;

  /**
   * Lista simplemente enlazada que tiene los componentes (número u operador)
   * de la expresión algebraica ya dividida.
   */
  var $Piezas = array();

  /**
   * Arreglo unidimensional con las 26 variables diferentes
   */
  var $valorVariable = array();

  /**
   * Almacena hasta que nivel se llega en paréntesis internos
   */
  var $MaximoNivel;

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  var $ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |8.3| |/| |5| |*| |7.4|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  public function Analizar($expr)
  {
    /* Inicializa la lista de piezas  */
    unset($this->Piezas);

    /* Inicializa el nivel */
    $nivel = 0;
    $this->MaximoNivel = 0;

    /* Tamaño de la expresión simple */
    $longitud = strlen($expr);

    /* Conversión de string a double */
    $entero = 0;
    $fraccion = 0;
    $divide = 1;
    $puntoentero = false;

    /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
    $acumulabaNumero = false;

    for ($cont=0; $cont<$longitud; $cont++)
    {
      /* Trae la letra */
      $letra = $expr{$cont};

      /* Si hay un paréntesis que abre, el nivel aumenta */
      if ($letra=='(')
      {
        $nivel++;
        if ($nivel>$this->MaximoNivel) $this->MaximoNivel = $nivel;
      }
      /* Si es paréntesis que cierra, el nivel disminuye */
      else if ($letra==')')
        $nivel--;
      /* Si es una variable o una función */
      else if ($letra >= 'a' && $letra <= 'z')
      {
          $objeto = new Pieza();
          $objeto->ConstructorVariable(ord($letra) - $this->ASCIILETRA, $nivel);
          $this->Piezas[] = $objeto;
```

```
    }
    /* Si es un número */
    else if( ($letra >= '0' && $letra <= '9') || $letra == '.')
    {
      /* Ir armando el número de tipo double */
      if ($letra == '.')
        $puntoentero = true;
      else
        if(!$puntoentero)  /* $puntoentero == false */
          $entero = $entero * 10 + $letra;
        else
        {
          $fraccion = $fraccion * 10 + $letra;
          $divide *=10;
        }
        $acumulabaNumero = true;
    }
    else
    {
      /* Si es un operador entonces crea el nodo del operador y el
      nodo del número si venía acumulando un número */
      if ($acumulabaNumero)
      {
        $objeto = new Pieza();
        $objeto->ConstructorNumero($entero + $fraccion/$divide, $nivel);
        $this->Piezas[] = $objeto;
      }

      /* Agrega a la lista */
      $objeto = new Pieza();
      $objeto->ConstructorOperador($letra, $nivel);
      $this->Piezas[] = $objeto;

      /* Inicializa de nuevo las variables de conversión de string a número */
      $entero = 0;
      $fraccion = 0;
      $divide = 1;
      $puntoentero = false;
      $acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if ($acumulabaNumero)
  {
    $objeto = new Pieza();
    $objeto->ConstructorNumero($entero + $fraccion/$divide, $nivel);
    $this->Piezas[] = $objeto;
  }
}

/**
 * Ya construida la lista del tipo:
 * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
  public function Evaluar()
  {
      $pos=0;
      $antes=0;
      $sigue=0;
      $this->ERRORMATEMATICO = false;

      /* Total de nodos en la lista creada */
      $totalPiezas = sizeof($this->Piezas);

      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
          /* Activa todas las piezas para que sean evaluadas */
          $this->Piezas[$pos]->setEvaluado(false);

          /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
          ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
          if ($this->Piezas[$pos]->getTipo() == $this->ESVARIABLE)
              $this->Piezas[$pos]->setAcumula($this->valorVariable[$this->Piezas[$pos]->getVariable()]);
          else if ($this->Piezas[$pos]->getTipo() == $this->ESNUMERO)
              $this->Piezas[$pos]->setAcumula($this->Piezas[$pos]->getNumero());
      }

      /* Va del nivel mas profundo al mas superficial */
      for ($evaluaNivel = $this->MaximoNivel; $evaluaNivel >= 0; $evaluaNivel--)
      {

      /* Recorre toda la lista */
      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
        /* Si encuentra un nodo del tipo operador y es exponente */
```

```
          if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && $this-
>Piezas[$pos]->getOperador() == '^')
          {
            /* Busca un nodo anterior que sea número y no haya sido evaluado */
            for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

            /* Busca un nodo siguiente que sea número y no haya sido evaluado */
            for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

            /* Marca esos nodos actual y siguiente como ya evaluados */
            $this->Piezas[$pos]->setEvaluado(true);
            $this->Piezas[$sigue]->setEvaluado(true);

            /* Hace la operación de Número elevado a Número */
            $this->Piezas[$antes]->setAcumula(pow($this->Piezas[$antes]->getAcumula(), $this->Piezas[$sigue]->getAcumula()));
          }
        }

        /* Recorre toda la lista */
        for ($pos=0; $pos<$totalPiezas; $pos++)
        {
          /* Si encuentra un nodo del tipo operador y es multiplicación o división */
          if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this-
>Piezas[$pos]->getOperador() == '*' || $this->Piezas[$pos]->getOperador() == '/'))
          {
            /* Busca un nodo anterior que sea número y no haya sido evaluado */
            for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

            /* Busca un nodo siguiente que sea número y no haya sido evaluado */
            for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

            /* Marca esos nodos actual y siguiente como ya evaluados */
            $this->Piezas[$pos]->setEvaluado(true);
            $this->Piezas[$sigue]->setEvaluado(true);

            /* Hace la operación de Número * Número, o Número / Número */
            if ($this->Piezas[$pos]->getOperador() == '*')
              $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() *  $this->Piezas[$sigue]->getAcumula() );
            else
            {
              if ($this->Piezas[$sigue]->getAcumula() != 0)
                $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() /  $this->Piezas[$sigue]->getAcumula() );
              else
              {
                $this->ERRORMATEMATICO = true;
                return 0;
              }
            }
          }
        }

        /* Recorre toda la lista */
        for ($pos=0; $pos<$totalPiezas; $pos++)
        {
            /* Si encuentra un nodo del tipo operador y es suma o resta */
            if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR &&
($this->Piezas[$pos]->getOperador() == '+' || $this->Piezas[$pos]->getOperador() == '-'))
            {
              /* Busca un nodo anterior que sea número y no haya sido evaluado */
              for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

              /* Busca un nodo siguiente que sea número y no haya sido evaluado */
              for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

              /* Marca esos nodos actual y siguiente como ya evaluados */
              $this->Piezas[$pos]->setEvaluado(true);
              $this->Piezas[$sigue]->setEvaluado(true);

              /* Hace la operación de Número + Número, o Número - Número */
              if ($this->Piezas[$pos]->getOperador() == '+')
                $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() +  $this->Piezas[$sigue]->getAcumula() );
              else
                $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() -  $this->Piezas[$sigue]->getAcumula() );
            }
        }
    }
    /* Resultado de la expresión */
    return $this->Piezas[0]->getAcumula();
  }

  /**
   * Guarda en un arreglo unidimensional el valor de las variables
   * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
   * @param valor  Valor de tipo double que tendrá esa variable.
   */
  public function DarValorVariable($variable, $valor)
  {
    $this->valorVariable[ord($variable) - $this->ASCIILETRA] = $valor;
  }

  /**
   * Retorna true si hubo un error matemático
```

```
   * @return TRUE si hubo un error matemático
   */
  public function getERRORMATEMATICO()
  {
      return $this->ERRORMATEMATICO;
  }
}
?>
```

index.php

```
<?php
include("EvaluadorExpresionAlgebraica.php");

$objEvaluador = new EvaluadorExpresionAlgebraica();

echo "Inicia con paréntesis<br>";

$expresion = '(x*5.103*(x+x/8.527*9.383))';
$objEvaluador->Analizar($expresion);
$objEvaluador->DarValorVariable('x', 13.1729);
$valor = $objEvaluador->Evaluar();
echo $valor;
?>
```

JAVASCRIPT

EvaluadorExpresionAlgebraica.js

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
function Pieza()
{
  /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
  this.nivel;

  /* 1. Almacena un número, 2. Almacena un operador */
  this.tipo;

  /* El número en la expresión algebraica */
  this.numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  this.operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  this.evaluado;

/* El resultado parcial de la evaluación de la expresión */
this.acumula;

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 */
this.ConstructorNumero = function(numero)
{
 this.tipo = 1; //Es un número
 this.numero = numero;

 this.evaluado = false;
 this.acumula = numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 */
this.ConstructorOperador = function(operador, nivel)
{
  this.tipo = 2; //Es un operador
  this.operador = operador;
  this.nivel = nivel;

  this.evaluado = false;
  this.acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
this.ConstructorVariable = function(variable, nivel)
{
 this.tipo = 3; //Es una variable
 this.variable = variable;
 this.nivel = nivel;

 this.evaluado = false;
 this.acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
this.getAcumula = function()
{
  return this.acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
this.isEvaluado = function()
{
  return this.evaluado;
}

/**
```

```
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
this.getNumero = function()
{
  return this.numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
this.getOperador = function()
{
  return this.operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
this.getVariable = function()
{
  return this.variable;
}

/**
 * Retorna que tipo de pieza es: número u operador
 * @return Tipo de pieza
 */
  this.getTipo = function()
  {
    return this.tipo;
  }

  /**
   * Retorna en que nivel se encuentra la pieza con respecto a la expresión
   * @return Nivel
   */
  this.getNivel = function()
  {
     return this.nivel;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  this.setAcumula = function(acumulado)
  {
    this.acumula = acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  this.setEvaluado = function(evaluado)
  {
    this.evaluado = evaluado;
  }
}

/*****************************************************************************************************
*****************************************************************************************************

*****************************************************************************************************
*/


function EvaluadorExpresionAlgebraica()
/**
 * Evaluador de expresiones algebraicas.
 * Recibe la expresión algebraica por ejemplo: "sin(x)+cos(y)*3.14"
 * como una cadena (String) junto con el valor de las variables (x=8.92, y=-7.43)
 * el algoritmo interpreta esa expresión respetando las reglas
 * algebraicas y retorna el resultado de tipo double.
 *
 * @author Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web:  http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha:  Mayo de 2011
 */
{
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    this.ESNUMERO = 1;
    this.ESOPERADOR = 2;
    this.ESVARIABLE = 3;
```

```
    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    this.ASCIILETRA = 97;

    /**
     * Lista simplemente enlazada que tiene los componentes (número, operador o función)
     * de la expresión algebraica ya dividida.
     */
    this.Piezas = [];

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    this.valorVariable = [];

    /**
     * Almacena hasta que nivel se llega en paréntesis internos
     */
    this.MaximoNivel;

    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    this.ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    this.Analizar = function(expr)
    {
        /* Inicializa la lista de piezas  */
        this.Piezas.length = 0;
        NumeroPieza = 0;

        /* Inicializa el nivel */
        nivel = 0;
        this.MaximoNivel = 0;

        /* Tamaño de la expresión simple */
        longitud = expr.length;

        /* Conversión de string a double */
        entero = 0;
        fraccion = 0;
        divide = 1;
        puntoentero = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        acumulabaNumero = false;

        for (cont=0; cont<longitud; cont++)
        {
            /* Trae la letra */
            letra = expr.charAt(cont);

            /* Si hay un paréntesis que abre, el nivel aumenta */
            if (letra=='(')
            {
                nivel++;
                if (nivel>this.MaximoNivel) this.MaximoNivel = nivel;
            }
            /* Si es paréntesis que cierra, el nivel disminuye */
            else if (letra==')')
                nivel--;
            /* Si es una variable o una función */
            else if (letra >= 'a' && letra <= 'z')
            {
                objeto = new Pieza();
                objeto.ConstructorVariable(letra - this.ASCIILETRA, nivel);
                this.Piezas[NumeroPieza++] = objeto;
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntoentero = true;
                else
                    if(!puntoentero)  /* puntoentero == false */
                        entero = entero * 10 + parseFloat(letra);
                    else
                    {
                        fraccion = fraccion * 10 + parseFloat(letra);
```

```
                    divide *=10;
                }
                acumulabaNumero = true;
        }
        else
        {
            /* Si es un operador entonces crea el nodo del operador y el
                nodo del número si venía acumulando un número */
            if (acumulabaNumero)
            {
                /* El nodo del número */
                objeto = new Pieza();
                objeto.ConstructorNumero(entero + fraccion/divide, nivel);
                this.Piezas[NumeroPieza++] = objeto;
            }

            /* El nodo operador */
            objeto = new Pieza();
            objeto.ConstructorOperador(letra, nivel);

            /* Agrega a la lista */
            this.Piezas[NumeroPieza++] = objeto;

            /* Inicializa de nuevo las variables de conversión de string a número */
            entero = fraccion = 0;
            divide = 1;
            puntoentero = acumulabaNumero = false;
        }
    }

    /* Cierra la expresión simple preguntando si el último operando es un número */
    if (acumulabaNumero)
    {
        objeto = new Pieza();
        objeto.ConstructorNumero(entero + fraccion/divide, nivel);
        this.Piezas[NumeroPieza++] = objeto;
    }
}

/**
 * Ya construida la lista del tipo:
 * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
 * es ir del operador con mas precedencia al de menos precedencia.
 *
 * Este método se llama después de haber sido analizada la expresión.
 *
 * En el caso que sólo cambia el valor de una variable, no es necesario
 * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
 * la misma expresión.
 *
 * @return El valor de la expresión evaluada (double)
 */
this.Evaluar = function()
{
    pos=0, antes=0, sigue=0;
    ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    totalPiezas = this.Piezas.length;

    for (pos=0; pos<totalPiezas; pos++)
    {
        /* Activa todas las piezas para que sean evaluadas */
        this.Piezas[pos].setEvaluado(false);

        /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
           ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
        if (this.Piezas[pos].getTipo() == this.ESVARIABLE)
            this.Piezas[pos].setAcumula(this.valorVariable[this.Piezas[pos].getVariable()]);
        else if (this.Piezas[pos].getTipo() == this.ESNUMERO)
            this.Piezas[pos].setAcumula(this.Piezas[pos].getNumero());
    }

    /* Va del nivel mas profundo al mas superficial */
    for (evaluaNivel = this.MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
    {
        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es exponente */
            if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
this.Piezas[pos].getOperador() == '^')
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                this.Piezas[pos].setEvaluado(true);
                this.Piezas[sigue].setEvaluado(true);
```

```
                    /* Hace la operación de Número elevado a Número */
                    this.Piezas[antes].setAcumula(Math.pow(this.Piezas[antes].getAcumula(), this.Piezas[sigue].getAcumula()));
                }
            }

            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es multiplicación o división */
                if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
(this.Piezas[pos].getOperador() == '*' || this.Piezas[pos].getOperador() == '/'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    this.Piezas[pos].setEvaluado(true);
                    this.Piezas[sigue].setEvaluado(true);

                    /* Hace la operación de Número * Número, o Número / Número */
                    if (this.Piezas[pos].getOperador() == '*')
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() *  this.Piezas[sigue].getAcumula() );
                    else
                    {
                        if (this.Piezas[sigue].getAcumula() != 0)
                            this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() /  this.Piezas[sigue].getAcumula() );
                        else
                        {
                            ERRORMATEMATICO = true;
                            return 0;
                        }
                    }

                }
            }

            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es suma o resta */
                if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
(this.Piezas[pos].getOperador() == '+' || this.Piezas[pos].getOperador() == '-'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    this.Piezas[pos].setEvaluado(true);
                    this.Piezas[sigue].setEvaluado(true);

                    /* Hace la operación de Número + Número, o Número - Número */
                    if (this.Piezas[pos].getOperador() == '+')
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() +  this.Piezas[sigue].getAcumula() );
                    else
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() -  this.Piezas[sigue].getAcumula() );
                }
            }
        }

        /* Resultado de la expresión */
        return this.Piezas[0].getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    this.DarValorVariable = function(variable, valor)
    {
         this.valorVariable[variable - this.ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    this.getErrorMatematico = function()
    {
            return ERRORMATEMATICO;
    }
}
```

index.html

```
<html>
  <body>
    <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
    <script type="text/javascript">
    /* Probando el evaluador de expresiones */
    objEvaluador = new EvaluadorExpresionAlgebraica();
    expresion = "(((x-8.736*9.26-8.492*8.277+(x*1.490*9.291)*1.445)))";
    objEvaluador.Analizar(expresion);
    objEvaluador.DarValorVariable('x', 13.1729);
    valor = objEvaluador.Evaluar();
    document.write(valor);
      </script>
   </body>
</html>
```

Object Pascal

Pieza.pas

```
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web: http://darwin.50webs.com
// Licencia: LGPL
// Fecha: Enero de 2012
//
unit Pieza;

interface
type
  TPieza = class
  private
    // Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    nivel: integer;

    // 1. Almacena un número, 2. Almacena un operador
    tipo: integer;

    // El número en la expresión algebraica */
    numero: double;

    // El operador (+, -, *, /, ^) en la expresión algebraica
    operador: char;

    // La variable
    variable: integer;

    //Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    //cálculos que determinan el valor de la expresión
    evaluado: boolean;

    //El resultado parcial de la evaluación de la expresión */
    acumula: double;

   public
     Constructor Create(numero: double; nivel: integer); overload;
     Constructor Create(operador: char; nivel: integer); overload;
     Constructor Create(variable: integer; nivel: integer); overload;
     function getAcumula(): double;
     function isEvaluado(): boolean;
     function getNumero(): double;
     function getOperador(): char;
     function getVariable(): integer;
     function getTipo(): integer;
     function getNivel(): integer;
     procedure setAcumula(acumulado: double);
     procedure setEvaluado(evaluado: boolean);
   end;
implementation


// Constructor en caso que la pieza contenga un número real
// @param numero Número de tipo double
// @param nivel Nivel en que se encuentra en la expresión este operador
constructor TPieza.Create(numero: double; nivel: integer);
begin
   self.tipo := 1; //Es un número
   self.numero := numero;
   self.nivel := nivel;

   self.evaluado := false;
   self.acumula := numero;
end;


// Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
// @param operador Que puede ser +, -, *, /, ^
// @param nivel Nivel en que se encuentra en la expresión este operador
constructor TPieza.Create(operador: char; nivel: integer);
begin
    self.tipo := 2; //Es un operador
    self.operador := operador;
    self.nivel := nivel;

    self.evaluado := false;
    self.acumula := 0;
end;


// Constructor en caso que el nodo contenga una variable
// @param variable Puede ir de 0 a 25 que representa de a..z
// @param nivel Nivel en que se encuentra en la expresión esta variable
constructor TPieza.Create(variable: integer; nivel: integer);
begin
     self.tipo := 3; //Es una variable
     self.variable := variable;
     self.nivel := nivel;
```

```
     self.evaluado := false;
     self.acumula := 0;
end;

// Retorna el acumulado que tiene esta pieza
// @return Acumulado de la pieza
function TPieza.getAcumula(): double;
begin
    getAcumula := acumula;
end;

// Retorna si la pieza ya fue evaluada
// @return True si la pieza ya fue evaluada
function TPieza.isEvaluado(): boolean;
begin
    isEvaluado := evaluado;
end;


// Retorna el número de tipo double que tiene la pieza
// @return El valor del número tipo double
function TPieza.getNumero(): double;
begin
    getNumero := numero;
end;

// Retorna el operador (+, -, *, /, ^) que tiene la pieza
// @return El operador en char
function TPieza.getOperador(): char;
begin
    getOperador := operador;
end;

// Retorna la variable que tiene la pieza
// @return La variable
function TPieza.getVariable(): integer;
begin
    getVariable := variable;
end;

// Retorna que tipo de pieza es: número u operador
// @return Tipo de pieza
function TPieza.getTipo(): integer;
begin
    getTipo := tipo;
end;

// Retorna en que nivel se encuentra la pieza con respecto a la expresión
// @return Nivel
function TPieza.getNivel(): integer;
begin
    getNivel := nivel;
end;

// Da valor al acumulado por pieza
// @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
procedure TPieza.setAcumula(acumulado:double);
begin
    self.acumula := acumulado;
end;

// Marca la pieza como ya evaluada
// @param evaluado true si la pieza ya ha sido evaluada, false si no
procedure TPieza.setEvaluado(evaluado: boolean);
begin
    self.evaluado := evaluado;
end;

end.
```

EvaluadorExpresionAlgebraica.pas

```
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web:  http://darwin.50webs.com
// Licencia: LGPL
// Fecha:  Marzo de 2012
unit EvaluadorExpresionAlgebraica;

interface
uses
  //Requerido para el TobjectList
  Contnrs, SysUtils, Math, Pieza;

type
  TEvaluadorExpresionAlgebraica = class
  private
      //Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
      ESNUMERO: integer;
      ESOPERADOR: integer;
      ESVARIABLE: integer;
```

```
    //Esta constante sirve para que se reste al carácter y se obtenga el número
    //Ejemplo:  '7' - ASCIINUMERO =  7
    ASCIINUMERO: integer;

    //Esta constante sirve para que se reste al carácter y se obtenga el número
    //usado en el arreglo unidimensional que alberga los valores de las variables
    //Ejemplo:  'b' - ASCIILETRA =  1
    ASCIILETRA: integer;

    //Lista simplemente enlazada que tiene los componentes (número u operador)
    //de la expresión algebraica ya dividida.
    Piezas: TobjectList;
    objPieza: TPieza;

    //Arreglo unidimensional con las 26 variables diferentes
    valorVariable: array[0..26] of double;

    //Almacena hasta que nivel se llega en paréntesis internos
    MaximoNivel: integer;

    //Variable que se pone a true cuando hay un error matemático
    ERRORMATEMATICO: boolean;
public
    Constructor Create();
    procedure Analizar(expr: string);
    function Evaluar(): double;
    function getErrorMatematico(): boolean;
    procedure DarValorVariable(variable: char; valor: double);
    procedure LiberaMemoria();
  end;

implementation

constructor TEvaluadorExpresionAlgebraica.Create();
begin
   self.ESNUMERO := 1;
   self.ESOPERADOR := 2;
   self.ESVARIABLE := 3;
   self.ASCIINUMERO := 48;
   self.ASCIILETRA := 97;
end;

//Este método se encarga de analizar la expresión y convertirla en una
//estructura que permita evaluar la expresión.
//
//Convierte una expresión algebraica en una sucesión de nodos.
//|2| |+| |8.3| |/| |5| |*| |7.4|
//
//@param expr La expresión algebraica sin espacios y en minúsculas

procedure TEvaluadorExpresionAlgebraica.Analizar(expr: string);
var
  longitud: integer;

  //Va extrayendo letra a letra de la expresión simple */
  letra: char;

  //Conversión de string a double */
  entero, fraccion, divide: integer;
  puntoentero: boolean;

  //Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero: boolean;

  cont: integer;

  //Nivel
  nivel: integer;

begin
  // Inicializa la lista de piezas
  Piezas := TObjectList.Create;

  // Tamaño de la expresión simple
  longitud := length(expr);

  // Conversión de string a double
  entero := 0;
  fraccion := 0;
  divide := 1;
  puntoentero := false;

  // Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero := false;

  //Inicializa el nivel
  nivel := 0;
  MaximoNivel := 0;

  for cont := 1 to longitud do
  begin
    //Trae la letra
```

```
    letra := expr[cont];

    if letra='(' then
    begin
      nivel := nivel + 1;
      if nivel > MaximoNivel then MaximoNivel := nivel;
    end
    else if letra = ')' then
      nivel := nivel - 1
    // Si es una variable
    else if letra in ['a'..'z'] then
    begin
      objPieza := TPieza.Create(ord(letra) - ASCIILETRA, nivel);
      Piezas.Add(objPieza);
    end
    // Si es un número
    else if (letra in ['0'..'9']) or (letra = '.') then
    begin
      //Ir armando el número de tipo double */
      if letra = '.' then
        puntoentero := true
      else
      begin
        if not(puntoentero) then // puntoentero = false
          entero := entero * 10 + ord(letra) - ASCIINUMERO
        else
        begin
          fraccion := fraccion * 10 + ord(letra) - ASCIINUMERO;
          divide := divide*10;
        end;
        acumulabaNumero := true;
      end
    end
    else
      begin
            // Si es un operador entonces crea el nodo del operador y el
            // nodo del número si venía acumulando un número */
            if acumulabaNumero then
            begin
              objPieza := TPieza.Create(entero + fraccion/divide, nivel);
              Piezas.Add(objPieza);
            end;

            // Agrega a la lista el operador */
            objPieza := TPieza.Create(letra, nivel);
            Piezas.Add(objPieza);

            // Inicializa de nuevo las variables de conversión de string a número */
            entero := 0;
            fraccion := 0;
            divide := 1;
            puntoentero := false;
            acumulabaNumero := false;
        end;
      end;

      // Cierra la expresión simple preguntando si el último operando es un número */
      if acumulabaNumero then
      begin
        objPieza := TPieza.Create(entero + fraccion/divide, nivel);
        Piezas.Add(objPieza);
      end;
    end;


    // Ya construida la lista del tipo:
    // [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
    // es ir del operador con mas precedencia al de menos precedencia.
    //
    // Este método se llama después de haber sido analizada la expresión.
    //
    // @return El valor de la expresión evaluada (double)
    function TEvaluadorExpresionAlgebraica.Evaluar(): double;
    var
      pos, antes, sigue, totalPiezas: integer;
      evaluanivel: integer;
    begin
      ERRORMATEMATICO := false;
    try
      // Total de nodos en la lista creada
      totalPiezas := Piezas.Count - 1;

      for pos := 0 to totalPiezas do
      begin
        //Activa todas las piezas para que sean evaluadas */
        (Piezas.Items[pos] as TPieza).setEvaluado(false);

        // Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
        //¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
        if (Piezas.Items[pos] as Tpieza).getTipo() = ESVARIABLE then
           (Piezas.Items[pos] as Tpieza).setAcumula(valorVariable[(Piezas.Items[pos] as Tpieza).getVariable()])
        else if (Piezas.Items[pos] as Tpieza).getTipo() = ESNUMERO then
```

```
      (Piezas.Items[pos] as Tpieza).setAcumula((Piezas.Items[pos] as Tpieza).getNumero());
  end;

  // Va del nivel mas profundo al mas superficial */
  for evaluaNivel := MaximoNivel downto 0 do
  begin
       // Recorre toda la lista
       for pos :=0 to totalPiezas do
       begin
           //Si encuentra un nodo del tipo operador y es exponente */
           if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and ((Piezas.Items[pos] as Tpieza).getOperador() = '^') then
           begin
             //Busca un nodo anterior que sea número y no haya sido evaluado */
             antes := pos-1;
             while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

             //Busca un nodo siguiente que sea número y no haya sido evaluado */
             sigue := pos + 1;
             while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

             // Marca esos nodos actual y siguiente como ya evaluados */
             (Piezas.Items[pos] as Tpieza).setEvaluado(true);
             (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

              // Hace la operación de Número elevado a Número
              (Piezas.Items[antes] as Tpieza).setAcumula(Power((Piezas.Items[antes] as Tpieza).getAcumula(),
(Piezas.Items[sigue] as Tpieza).getAcumula()));
           end;
       end;

       // Recorre toda la lista
       for pos :=0 to totalPiezas do
       begin
           // Si encuentra un nodo del tipo operador y es multiplicación o división */
           if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '*') or ((Piezas.Items[pos] as Tpieza).getOperador() = '/'))
then
           begin
             //Busca un nodo anterior que sea número y no haya sido evaluado */
             antes := pos-1;
             while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

             //Busca un nodo siguiente que sea número y no haya sido evaluado */
             sigue := pos + 1;
             while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

             // Marca esos nodos actual y siguiente como ya evaluados */
             (Piezas.Items[pos] as Tpieza).setEvaluado(true);
             (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

             // Hace la operación de Número * Número, o Número / Número */
             if (Piezas.Items[pos] as Tpieza).getOperador() = '*' then
                 (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() *
(Piezas.Items[sigue] as Tpieza).getAcumula() )
             else
             begin
                 if (Piezas.Items[sigue] as Tpieza).getAcumula() <> 0 then
                   (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() /
(Piezas.Items[sigue] as Tpieza).getAcumula() )
                 else
                 begin
                   ERRORMATEMATICO := true;
                   Evaluar := 0;
                   Exit;
                 end;
             end;
           end;
       end;

       // Recorre toda la lista
       for pos :=0 to totalPiezas do
       begin
           // Si encuentra un nodo del tipo operador y es suma o resta */
           if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '+') or ((Piezas.Items[pos] as Tpieza).getOperador() = '-'))
then
           begin
             //Busca un nodo anterior que sea número y no haya sido evaluado */
             antes := pos-1;
             while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

             //Busca un nodo siguiente que sea número y no haya sido evaluado */
             sigue := pos + 1;
             while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

             // Marca esos nodos actual y siguiente como ya evaluados */
             (Piezas.Items[pos] as Tpieza).setEvaluado(true);
             (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

             // Hace la operación de Número + Número, o Número - Número */
             if (Piezas.Items[pos] as Tpieza).getOperador() = '+' then
```

```
                    (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() +
(Piezas.Items[sigue] as Tpieza).getAcumula() )
                else
                    (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() -
(Piezas.Items[sigue] as Tpieza).getAcumula() );
              end;
          end;
    end;

    // Resultado de la expresión
    Evaluar := (Piezas.Items[0] as Tpieza).getAcumula();

except
    on E : Exception do
    begin
      ERRORMATEMATICO := true;
      Evaluar := 0;
      exit;
    end;
end;

end;

// Guarda en un arreglo unidimensional el valor de las variables
// @param variable ¿Qué variable de la expresión recibirá el valor? a..z
// @param valor  Valor de tipo double que tendrá esa variable.
procedure TEvaluadorExpresionAlgebraica.DarValorVariable(variable: char; valor: double);
begin
          valorVariable[ord(variable) - ASCIILETRA] := valor;
end;

// Retorna true si hubo un error matemático
// @return TRUE si hubo un error matemático
function TEvaluadorExpresionAlgebraica.getErrorMatematico(): boolean;
begin
    getErrorMatematico := ERRORMATEMATICO;
end;

//Libera la memoria
procedure TEvaluadorExpresionAlgebraica.LiberaMemoria();
begin
    Piezas.Free;
end;

end.
```

PARTE 5. EVALUADOR COMPLETO. INTERPRETACIÓN DE FUNCIONES ALGEBRAICAS.

Cuando se hace uso de funciones en la expresión algebraica como seno, coseno, tangente, raizcuadrada, arcococoseno, arcoseno, etc., para interpretar esta nueva funcionalidad y que la interpretación y ejecución fueran rápidas, se puso en consideración:

1. Las funciones deben tener tres(3) letras: sin, sen, cos, tan, log
2. Después de una función sigue un paréntesis que abre. Por ejemplo: 4-cos(x+5)

Lo que hace el analizador de la expresión es que al ir de carácter en carácter en la expresión, al detectar una letra de la 'a' a la 'z', puede suceder que o es una función o es una variable. Si el carácter que sigue es una letra de la 'a' a la 'z' también, entonces estamos ante una función. En ese caso el analizador, trae la tercera letra y chequea con la lista de funciones soportadas, una vez detecta la función, crea una pieza de tipo función y adicionalmente la pieza sabe qué función es. Luego pieza tiene un nuevo atributo: "funcion" de tipo entero, y es el que indica que función es.

Al momento de evaluar la expresión, como se recorre el arraylist de piezas, a llegar a una pieza de tipo función, entonces se evalúa la función como tal.

C++

Pieza.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */

/* Pieza será la unidad básica de la expresión algebraica, cada nodo tendrá o un operador
   o un número o una variable. Este nodo trabaja en una lista simplemente enlazada */
class Pieza
{
private:
    /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    int nivel;

    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable */
    int tipo;

    /* El número en la expresión algebraica */
    double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    char operador;

    /* La variable */
    int variable;

    /* La función */
    int funcion;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    bool evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    double acumula;

public:
    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     * @param nivel Nivel en que se encuentra en la expresión este número
     */
    Pieza(double, int);

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    Pieza(char, int);

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    Pieza(int, int);

    /**
     * Constructor en caso que la pieza contenga una función
     * @param funcion Identificación de la función
     * @param nivel Nivel en que se encuentra en la expresión este operador
     * @param bandera Sólo sirve para diferenciarlo del anterior constructor
     */
    Pieza(int, int, char);

    /** Retorna el acumulado que tiene esta pieza
```

```
     * @return Acumulado de la pieza
     */
    double getAcumula();

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    bool isEvaluado();

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    double getNumero();

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    char getOperador();

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    int getVariable();

    /**
     * Retorna que tipo de pieza es: número, operador, variable o función
     * @return Tipo de pieza
     */
    int getTipo();

    /**
     * Retorna en qué nivel se encuentra la pieza con respecto a la expresión
     * @return Nivel
     */
    int getNivel();

    /**
     * Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
     * @return Código de la función
     */
    int getFuncion();

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    void setAcumula(double);

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(bool);
};
```

Pieza.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include <stdio.h>
#include <math.h>
#include "Pieza.h"

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 * @param nivel Nivel en que se encuentra en la expresión este número
 */
Pieza::Pieza(double numero, int nivel)
{
 this->tipo = 1; //Es un número
 this->numero = numero;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
Pieza::Pieza(char operador, int nivel)
```

```
{
 this->tipo = 2; //Es un operador
 this->operador = operador;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
Pieza::Pieza(int variable, int nivel)
{
 this->tipo = 3; //Es una variable
 this->variable = variable;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = 0;
}

/**
 * Constructor en caso que la pieza contenga una función
 * @param funcion Identificación de la función
 * @param nivel Nivel en que se encuentra en la expresión este operador
 * @param bandera Sólo sirve para diferenciarlo del anterior constructor
 */
Pieza::Pieza(int funcion, int nivel, char bandera)
{
 this->tipo = 4; //Es una función
 this->funcion = funcion;
 this->nivel = nivel;

 this->evaluado = false;
 this->acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
 */
double Pieza::getAcumula()
{
 return acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
bool Pieza::isEvaluado()
{
 return evaluado;
}

/**
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
double Pieza::getNumero()
{
 return numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
char Pieza::getOperador()
{
 return operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
int Pieza::getVariable()
{
 return variable;
}

/**
 * Retorna que tipo de pieza es: número, operador, variable o función
 * @return Tipo de pieza
 */
int Pieza::getTipo()
{
 return tipo;
}
```

```
/**
 * Retorna en qué nivel se encuentra la pieza con respecto a la expresión
 * @return Nivel
 */
int Pieza::getNivel()
{
    return nivel;
}

/**
 * Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
 * @return Código de la función
 */
int Pieza::getFuncion()
{
    return funcion;
}

/**
 * Da valor al acumulado por pieza
 * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
 */
void Pieza::setAcumula(double acumulado)
{
 this->acumula = acumulado;
}

/**
 * Marca la pieza como ya evaluada
 * @param evaluado true si la pieza ya ha sido evaluada, false si no
 */
void Pieza::setEvaluado(bool evaluado)
{
 this->evaluado = evaluado;
}
```

EvaluadorExpresionAlgebraica.h

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include "Pieza.h"
#include <vector>

class EvaluadorExpresionAlgebraica
{
private:
    /**
     * Constantes para determinar qué tipo es cada pieza en que se divide la expresión algebraica
     */
    static const int ESNUMERO = 1;
    static const int ESOPERADOR = 2;
    static const int ESVARIABLE = 3;
    static const int ESFUNCION = 4;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    static const int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    static const int ASCIILETRA = 97;

    /**
     * Las funciones que soporta este evaluador
     */
    static const int TAMANOFUNCION = 39;
    char listaFunciones[50];

    /* Lista simplemente enlazada que tiene los componentes (numero u operador
       de la expresión algebraica ya dividida */
    std::vector<Pieza> Piezas;

    /* Arreglo unidimensional con las 26 variables diferentes */
    double valorVariable[30];

    /* Almacena hasta que nivel se llega en paréntesis internos */
    int MaximoNivel;

    /* Variable que se pone a true cuando hay error matemático */
    bool ERRORMATEMATICO;

public:
```

```
    EvaluadorExpresionAlgebraica(void);
    void DarValorVariable(char, double);

    /* Analiza la expresión algebraica y la vuelve una lista */
    void Analizar(char *);

    /* Evalúa la expresión que ha sido previamente analizada */
    double Evaluar(void);

    /* Retorna si hubo error matemático */
    bool getErrorMatematico();
};
```

EvaluadorExpresionAlgebraica.cpp

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include <math.h>
#include <stdio.h>

#include "EvaluadorExpresionAlgebraica.h"

/**
* Constructor
*/
EvaluadorExpresionAlgebraica::EvaluadorExpresionAlgebraica()
{
  char *funciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
  for (int cont = 0; *(funciones+cont); cont++)
      listaFunciones[cont] = funciones[cont];
}

/**
 * Convierte una expresión simple en una sucesión de nodos.
 * |2| |+| |a| |/| |5| |*| |c| |-| |1.5|
*/
void EvaluadorExpresionAlgebraica::Analizar(char *expr)
{
  /* Inicializa la lista de piezas. http://www.codeguru.com/Cpp/Cpp/cpp_mfc/stl/article.php/c4027  */
  Piezas.clear();

  /* Inicializa el nivel */
  int nivel = 0;
  MaximoNivel = 0;

  /* Conversión de string a double */
  int decimal = 0;
  int fraccion = 0;
  int divide = 1;
  bool puntodecimal = false;

  /* Tamaño de la expresión simple */
  int longitud = strlen(expr);

  /* Va extrayendo letra a letra de la expresión simple */
  char letra;

  /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
  bool acumulabaNumero = false;

  for (int cont=0; cont<longitud; cont++)
  {
    /* Trae la letra */
    letra = expr[cont];

    /* Si hay un paréntesis que abre, el nivel aumenta */
    if (letra=='(')
    {
      nivel++;
      if (nivel>MaximoNivel) MaximoNivel = nivel;
    }
    /* Si es paréntesis que cierra, el nivel disminuye */
    else if (letra==')')
      nivel--;
    /* Si es una variable o una función */
    else if (letra >= 'a' && letra <= 'z')
    {
      /* Detecta si es una función porque tiene dos letras seguidas */
      if (cont<longitud+1)
      {
        /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
        char letra2 = expr[cont+1];
        if ( letra2 >= 'a' && letra2 <= 'z')
        {
          char letra3 = expr[cont+2];

          /* Identifica las funciones */
```

```
        int funcionDetectada = 1;
        for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
        {
          if (letra == listaFunciones[funcion]
            && letra2 == listaFunciones[funcion + 1]
            && letra3 == listaFunciones[funcion + 2])
            break;
          funcionDetectada++;
        }

        /* Adiciona a la lista */
        Pieza objeto(funcionDetectada, nivel, 'f');
        Piezas.push_back(objeto);

        nivel++;
        if (nivel>MaximoNivel) MaximoNivel = nivel;

        /* Mueve tres caracteres  sin(  [s][i][n][(] */
        cont+=3;
      }
      /* Es una variable, no una función */
      else
      {
        Pieza objeto(letra - ASCIILETRA, nivel);
        Piezas.push_back(objeto);
      }
    }
    /* Es una variable, no una función */
    else
    {
      Pieza objeto(letra - ASCIILETRA, nivel);
      Piezas.push_back(objeto);
    }
  }
  /* Si es un número */
  else if( (letra >= '0' && letra <= '9') || letra == '.')
  {
    /* Ir armando el número de tipo double */
    if (letra == '.')
      puntodecimal = true;
    else
      if(!puntodecimal)  /* puntodecimal == false */
        decimal = decimal * 10 + letra-ASCIINUMERO;
      else
      {
        fraccion = fraccion * 10 + letra-ASCIINUMERO;
        divide *=10;
      }
    acumulabaNumero = true;
  }
  else
  {
    /* Si es un operador entonces crea el nodo del operador y el
       nodo del número si venía acumulando un número */
    if (acumulabaNumero)
    {
      /* El nodo del número */
      Pieza objeto(decimal + (double) fraccion/divide, nivel);

      /* Agrega a la lista */
      Piezas.push_back(objeto);
    }

    /* El nodo operador */
      /* El nodo operador */
      Pieza objeto(letra, nivel);

      /* Agrega a la lista */
      Piezas.push_back(objeto);

      /* Inicializa de nuevo las variables de conversión de string a número */
      decimal = 0;
      fraccion = 0;
      divide = 1;
      puntodecimal = false;
      acumulabaNumero = false;
    }
  }

  /* Cierra la expresión simple preguntando si el último operando es un número */
  if (acumulabaNumero)
  {
    Pieza objeto(decimal + (double) fraccion/divide, nivel);
    Piezas.push_back(objeto);
  }

}

/* Ya construida la lista del tipo:
   [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   es ir del operador con mas precedencia al de menos precedencia */
double EvaluadorExpresionAlgebraica::Evaluar()
{
```

```
int pos=0, antes=0, sigue=0;
ERRORMATEMATICO = false;

/* Total de nodos en la lista creada */
int totalPiezas = Piezas.size();

for (pos=0; pos<totalPiezas; pos++)
{
  /* Activa todas las piezas para que sean evaluadas */
  Piezas[pos].setEvaluado(false);

  /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
    ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
  if (Piezas[pos].getTipo() == ESVARIABLE)
    Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
  else if (Piezas[pos].getTipo() == ESNUMERO)
    Piezas[pos].setAcumula(Piezas[pos].getNumero());
}

/* Va del nivel mas profundo al mas superficial */
for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
{
  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra una función la evalúa con la siguiente */
    if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo()==ESFUNCION )
    {
        switch (Piezas[pos].getFuncion())
{
    switch (Piezas[pos].getFuncion())
    {
      case 1:
      case 2:
        Piezas[pos].setAcumula(sin(Piezas[pos+1].getAcumula()));
        break;
      case 3:
        Piezas[pos].setAcumula(cos(Piezas[pos+1].getAcumula()));
        break;
      case 4:
        Piezas[pos].setAcumula(tan(Piezas[pos+1].getAcumula()));
        break;
      case 5:
        if (Piezas[pos+1].getAcumula() > 0)
          Piezas[pos].setAcumula(Piezas[pos+1].getAcumula());
        else
          Piezas[pos].setAcumula(-Piezas[pos+1].getAcumula());
        break;
      case 6:
        if (Piezas[pos+1].getAcumula() >= -1 && Piezas[pos+1].getAcumula() <= 1)
          Piezas[pos].setAcumula(asin(Piezas[pos+1].getAcumula()));
        else
        {
          ERRORMATEMATICO = true;
          return 0;
        }
        break;
      case 7:
        if (Piezas[pos+1].getAcumula() >= -1 && Piezas[pos+1].getAcumula() <= 1)
          Piezas[pos].setAcumula(acos(Piezas[pos+1].getAcumula()));
        else
        {
          ERRORMATEMATICO = true;
          return 0;
        }
        break;
      case 8:
        Piezas[pos].setAcumula(atan(Piezas[pos+1].getAcumula()));
        break;
      case 9:
        Piezas[pos].setAcumula(log(Piezas[pos+1].getAcumula()));
        break;
      case 10:
        Piezas[pos].setAcumula(ceil(Piezas[pos+1].getAcumula()));
        break;
      case 11:
        Piezas[pos].setAcumula(exp(Piezas[pos+1].getAcumula()));
        break;
      case 12:
        if (Piezas[pos+1].getAcumula() >= 0)
          Piezas[pos].setAcumula(sqrt(Piezas[pos+1].getAcumula()));
        else
        {
          ERRORMATEMATICO = true;
          return 0;
        }
        break;
      case 13:
        Piezas[pos].setAcumula(pow(Piezas[pos+1].getAcumula(), (double) 1 / 3));
        break;
    }

    /* Marca el nodo siguiente como ya evaluado */
    Piezas[pos+1].setEvaluado(true);
```

```
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es exponente */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número elevado a Número */
        Piezas[antes].setAcumula(pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es multiplicación o división */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*' ||
Piezas[pos].getOperador() == '/'))
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número * Número, o Número / Número */
        if (Piezas[pos].getOperador() == '*')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
        else
        {
          if (Piezas[sigue].getAcumula() != 0)
            Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
        }
      }
    }

    /* Recorre toda la lista */
    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Si encuentra un nodo del tipo operador y es suma o resta */
      if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+' ||
Piezas[pos].getOperador() == '-'))
      {
        /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
        for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

        /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
        for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

        /* Marca esos nodos actual y siguiente como ya evaluados */
        Piezas[pos].setEvaluado(true);
        Piezas[sigue].setEvaluado(true);

        /* Hace la operación de Número + Número, o Número - Número */
        if (Piezas[pos].getOperador() == '+')
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
        else
          Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
      }
    }
  }

  /* Resultado de la expresión */
  return Piezas[0].getAcumula();
}

/* Guarda en un arreglo unidimensional el valor de las variables */
void EvaluadorExpresionAlgebraica::DarValorVariable(char variable, double valor)
{
   valorVariable[variable - ASCIILETRA] = valor;
}
```

```
/* Retorna true si hubo un error matemático */
bool EvaluadorExpresionAlgebraica::getErrorMatematico()
{
  return ERRORMATEMATICO;
}
```

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "EvaluadorExpresionAlgebraica.h"

int main(void);

int main()
{
    char expresion[500];
    EvaluadorExpresionAlgebraica objEvaluador;
    double valor;

    strcpy(expresion, "4.763*5.704+(x+sen(3.670/cos(sen(atn(sen(4.877*cos(7.330)))))))");
    objEvaluador.Analizar(expresion);
    objEvaluador.DarValorVariable('x', 13.1729);
    valor=objEvaluador.Evaluar();
    printf("%f\n", valor);
    getchar();
    return 0;
}
```

Visual C# .NET

Pieza.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    public class Pieza
    {
        /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
        private int nivel;

        /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
        private int tipo;

        /* El número en la expresión algebraica */
        private double numero;

        /* El operador (+, -, *, /, ^) en la expresión algebraica */
        private char operador;

        /* La variable */
        private int variable;

        /* La función */
        private int funcion;

        /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
           cálculos que determinan el valor de la expresión */
        private bool evaluado;

        /* El resultado parcial de la evaluación de la expresión */
        private double acumula;

        /**
         * Constructor en caso que la pieza contenga un número real
         * @param numero Número de tipo double
         * @param nivel Nivel en que se encuentra en la expresión este número
         */
        public Pieza(double numero, int nivel)
        {
            this.tipo = 1; //Es un número
            this.numero = numero;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = numero;
        }

        /**
         * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
         * @param operador Que puede ser +, -, *, /, ^
         * @param nivel Nivel en que se encuentra en la expresión este operador
         */
        public Pieza(char operador, int nivel)
        {
            this.tipo = 2; //Es un operador
            this.operador = operador;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = 0;
        }

        /**
         * Constructor en caso que el nodo contenga una variable
         * @param variable Puede ir de 0 a 25 que representa de a..z
         * @param nivel Nivel en que se encuentra en la expresión este operador
         */
        public Pieza(int variable, int nivel)
        {
            this.tipo = 3; //Es una variable
            this.variable = variable;
            this.nivel = nivel;

            this.evaluado = false;
            this.acumula = 0;
        }

        /**
```

```
     * Constructor en caso que la pieza contenga una función
     * @param funcion Identificación de la función
     * @param nivel Nivel en que se encuentra en la expresión este operador
     * @param bandera Sólo sirve para diferenciarlo del anterior constructor
     */
    public Pieza(int funcion, int nivel, char bandera)
    {
        this.tipo = 4; //Es una función
        this.funcion = funcion;
        this.nivel = nivel;

        this.evaluado = false;
        this.acumula = 0;
    }

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
    public double getAcumula()
    {
        return acumula;
    }

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    public bool isEvaluado()
    {
        return evaluado;
    }

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    public double getNumero()
    {
        return numero;
    }

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    public char getOperador()
    {
        return operador;
    }

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    public int getVariable()
    {
        return variable;
    }

    /**
     * Retorna que tipo de pieza es: número, operador, variable o función
     * @return Tipo de pieza
     */
    public int getTipo()
    {
        return tipo;
    }

    /**
     * Retorna en que nivel se encuentra la pieza con respecto a la expresión
     * @return Nivel
     */
    public int getNivel()
    {
        return nivel;
    }

    /**
     * Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
     * @return Código de la función
     */
    public int getFuncion()
    {
        return funcion;
    }

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    public void setAcumula(double acumulado)
    {
        this.acumula = acumulado;
```

```
        }

         /**
         * Marca la pieza como ya evaluada
         * @param evaluado true si la pieza ya ha sido evaluada, false si no
         */
        public void setEvaluado(bool evaluado)
        {
            this.evaluado = evaluado;
        }
    }
}
```

EvaluadorExpresionAlgebraica.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
  class EvaluadorExpresionAlgebraica
  {
    /**
     * Constantes para determinar qué tipo es cada pieza en que se divide la expresión algebraica
     */
    private static int ESNUMERO = 1;
    private static int ESOPERADOR = 2;
    private static int ESVARIABLE = 3;
    private static int ESFUNCION = 4;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * Ejemplo:  '7' - ASCIINUMERO =  7
     */
    private static int ASCIINUMERO = 48;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    private static int ASCIILETRA = 97;

    /**
     * Las funciones que soporta este evaluador
     */
    private static int TAMANOFUNCION = 39;
    private static String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

    /**
     * Lista simplemente enlazada que tiene los componentes (número, operador o función)
     * de la expresión algebraica ya dividida.
     */
    private List<Pieza> Piezas = new List<Pieza>();

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    private double[] valorVariable = new double[26];

    /**
     * Almacena hasta que nivel se llega en paréntesis internos
     */
    private int MaximoNivel;

    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    private bool ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    public void Analizar(String expr)
    {
      /* Inicializa la lista de piezas  */
      Piezas.Clear();
```

```
/* Inicializa el nivel */
int nivel = 0;
MaximoNivel = 0;

/* Tamaño de la expresión simple */
int longitud = expr.Length;

/* Va extrayendo letra a letra de la expresión simple */
char letra;

/* Conversión de string a double */
int entero = 0, fraccion = 0, divide = 1;
bool puntoentero = false;

/* Si llevaba acumulando un valor numérico esta variable se vuelve true */
bool acumulabaNumero = false;

for (int cont=0; cont<longitud; cont++)
{
  /* Trae la letra */
  letra = expr[cont];

  /* Si hay un paréntesis que abre, el nivel aumenta */
  if (letra=='(')
  {
    nivel++;
    if (nivel>MaximoNivel) MaximoNivel = nivel;
  }
/* Si es paréntesis que cierra, el nivel disminuye */
else if (letra==')')
  nivel--;
/* Si es una variable o una función */
else if (letra >= 'a' && letra <= 'z')
{
  /* Detecta si es una función porque tiene dos letras seguidas */
  if (cont < longitud-1)
  {
    /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
    char letra2 = expr[cont+1];
    if ( letra2 >= 'a' && letra2 <= 'z')
    {
      char letra3 = expr[cont+2];

      /* Identifica las funciones */
      int funcionDetectada = 1;
      for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
      {
        if (letra == listaFunciones[funcion]
          && letra2 == listaFunciones[funcion + 1]
          && letra3 == listaFunciones[funcion + 2])
        break;
        funcionDetectada++;
      }

      /* Adiciona a la lista */
      Pieza objeto = new Pieza(funcionDetectada, nivel, 'f');
      Piezas.Add(objeto);

      nivel++;
      if (nivel>MaximoNivel) MaximoNivel = nivel;

      /* Mueve tres caracteres  sin(  [s][i][n][(] */
      cont+=3;
    }
    /* Es una variable, no una función */
    else
    {
      Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
      Piezas.Add(objeto);
    }
  }
  /* Es una variable, no una función */
  else
  {
    Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
    Piezas.Add(objeto);
  }
}
/* Si es un número */
else if( (letra >= '0' && letra <= '9') || letra == '.')
{
  /* Ir armando el número de tipo double */
  if (letra == '.')
    puntoentero = true;
  else
    if(!puntoentero)  /* puntoentero == false */
      entero = entero * 10 + letra-ASCIINUMERO;
    else
    {
      fraccion = fraccion * 10 + letra-ASCIINUMERO;
      divide *=10;
```

```
        }
        acumulabaNumero = true;
      }
      else
      {
        /* Si es un operador entonces crea el nodo del operador y el
          nodo del número si venía acumulando un número */
        if (acumulabaNumero)
        {
          /* El nodo del número */
          Pieza objeto = new Pieza(entero + (double) fraccion/divide, nivel);

          /* Agrega a la lista */
          Piezas.Add(objeto);
        }

        /* El nodo operador */
        Pieza objeto2 = new Pieza(letra, nivel);

        /* Agrega a la lista */
        Piezas.Add(objeto2);

        /* Inicializa de nuevo las variables de conversión de string a número */
        entero = fraccion = 0;
        divide = 1;
        puntoentero = acumulabaNumero = false;
      }
    }

    /* Cierra la expresión simple preguntando si el último operando es un número */
    if (acumulabaNumero)
    {
      Pieza objeto = new Pieza(entero + (double) fraccion/divide, nivel);
      Piezas.Add(objeto);
    }
  }

  /**
   * Ya construida la lista del tipo:
   * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   * es ir del operador con más precedencia al de menos precedencia.
   *
   * Este método se llama después de haber sido analizada la expresión.
   *
   * En el caso que sólo cambia el valor de una variable, no es necesario
   * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
   * la misma expresión.
   *
   * @return El valor de la expresión evaluada (double)
   */
  public double Evaluar()
  {
    int pos=0, antes=0, sigue=0;
    ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    int totalPiezas = Piezas.Count;

    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Activa todas las piezas para que sean evaluadas */
      Piezas[pos].setEvaluado(false);

      /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
         ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
      if (Piezas[pos].getTipo() == ESVARIABLE)
        Piezas[pos].setAcumula(valorVariable[Piezas[pos].getVariable()]);
      else if (Piezas[pos].getTipo() == ESNUMERO)
        Piezas[pos].setAcumula(Piezas[pos].getNumero());
    }

    /* Va del nivel más profundo al más superficial */
    for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
    {
      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra una pieza de tipo función la evalúa con el valor de la siguiente pieza */
        if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESFUNCION)
        {
          switch (Piezas[pos].getFuncion())
          {
            case 1:
            case 2:
              Piezas[pos].setAcumula(Math.Sin(Piezas[pos+1].getAcumula()));
              break;
            case 3:
              Piezas[pos].setAcumula(Math.Cos(Piezas[pos+1].getAcumula()));
              break;
            case 4:
              Piezas[pos].setAcumula(Math.Tan(Piezas[pos+1].getAcumula()));
              break;
```

```
            case 5:
              if (Piezas[pos+1].getAcumula() > 0)
                Piezas[pos].setAcumula(Piezas[pos+1].getAcumula());
              else
                Piezas[pos].setAcumula(-Piezas[pos+1].getAcumula());
              break;
            case 6:
              if (Piezas[pos+1].getAcumula() >= -1 && Piezas[pos+1].getAcumula() <= 1)
                Piezas[pos].setAcumula(Math.Asin(Piezas[pos+1].getAcumula()));
              else
              {
                ERRORMATEMATICO = true;
                return 0;
              }
              break;
            case 7:
              if (Piezas[pos+1].getAcumula() >= -1 && Piezas[pos+1].getAcumula() <= 1)
                Piezas[pos].setAcumula(Math.Acos(Piezas[pos+1].getAcumula()));
              else
              {
                ERRORMATEMATICO = true;
                return 0;
              }
              break;
            case 8:
              Piezas[pos].setAcumula(Math.Atan(Piezas[pos+1].getAcumula()));
              break;
            case 9:
              Piezas[pos].setAcumula(Math.Log(Piezas[pos+1].getAcumula()));
              break;
            case 10:
              Piezas[pos].setAcumula(Math.Ceiling(Piezas[pos+1].getAcumula()));
              break;
            case 11:
              Piezas[pos].setAcumula(Math.Exp(Piezas[pos+1].getAcumula()));
              break;
            case 12:
              if (Piezas[pos+1].getAcumula() >= 0)
                Piezas[pos].setAcumula(Math.Sqrt(Piezas[pos+1].getAcumula()));
              else
              {
                ERRORMATEMATICO = true;
                return 0;
              }
              break;
            case 13:
              Piezas[pos].setAcumula(Math.Pow(Piezas[pos+1].getAcumula(), (double)1 / 3));
              break;
          }

          /* Marca el nodo siguiente como ya evaluado */
          Piezas[pos+1].setEvaluado(true);
        }
      }

      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra un nodo del tipo operador y es exponente */
        if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && Piezas[pos].getOperador() == '^')
        {
          /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
          for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
          for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas[pos].setEvaluado(true);
          Piezas[sigue].setEvaluado(true);

          /* Hace la operación de Número elevado a Número */
          Piezas[antes].setAcumula(Math.Pow(Piezas[antes].getAcumula(), Piezas[sigue].getAcumula()));
        }
      }

      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra un nodo del tipo operador y es multiplicación o división */
        if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '*'
|| Piezas[pos].getOperador() == '/'))
        {
          /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
          for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
          for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas[pos].setEvaluado(true);
          Piezas[sigue].setEvaluado(true);
```

```
            /* Hace la operación de Número * Número, o Número / Número */
            if (Piezas[pos].getOperador() == '*')
              Piezas[antes].setAcumula( Piezas[antes].getAcumula() *  Piezas[sigue].getAcumula() );
            else
            {
              if (Piezas[sigue].getAcumula() != 0)
                Piezas[antes].setAcumula( Piezas[antes].getAcumula() /  Piezas[sigue].getAcumula() );
              else
              {
                ERRORMATEMATICO = true;
                return 0;
              }
            }

          }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
          /* Si encuentra un nodo del tipo operador y es suma o resta */
          if (Piezas[pos].getNivel() == evaluaNivel && Piezas[pos].getTipo() == ESOPERADOR && (Piezas[pos].getOperador() == '+'
|| Piezas[pos].getOperador() == '-'))
          {
            /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
            for (antes=pos-1; Piezas[antes].isEvaluado(); antes--);

            /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
            for (sigue=pos+1; Piezas[sigue].isEvaluado(); sigue++);

            /* Marca esos nodos actual y siguiente como ya evaluados */
            Piezas[pos].setEvaluado(true);
            Piezas[sigue].setEvaluado(true);

            /* Hace la operación de Número + Número, o Número - Número */
            if (Piezas[pos].getOperador() == '+')
              Piezas[antes].setAcumula( Piezas[antes].getAcumula() +  Piezas[sigue].getAcumula() );
            else
              Piezas[antes].setAcumula( Piezas[antes].getAcumula() -  Piezas[sigue].getAcumula() );
          }
        }
      }

      /* Retorna el resultado */
      return Piezas[0].getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    public void DarValorVariable(char variable, double valor)
    {
       valorVariable[variable - ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    public bool getErrorMatematico()
    {
        return ERRORMATEMATICO;
    }
  }
}
```

Program.cs

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EvaluadorCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
            string expresion;
```

```
            double valor;

            expresion = "x*9.247*acs(7.428)-x*1.39/sen(x)";
            objEvaluador.Analizar(expresion);
            objEvaluador.DarValorVariable('x', 13.1729);
            valor = objEvaluador.Evaluar();
            Console.WriteLine(valor);
            Console.ReadKey();
        }
    }
}
```

Visual Basic .NET

Pieza.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web: http://darwin.50webs.com
'Licencia: LGPL
'Fecha: Enero de 2012

Public Class Pieza
    'Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión
    Private nivel As Integer

    '1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función
    Private tipo As Integer

    'El número en la expresión algebraica
    Private numero As Double

    'El operador (+, -, *, /, ^) en la expresión algebraica
    Private operador As Char

    'La variable
    Private variable As Integer

    'La función
    Private funcion As Integer

    'Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    'cálculos que determinan el valor de la expresión
    Private evaluado As Boolean

    'El resultado parcial de la evaluación de la expresión
        Private acumula As Double

    'Constructor en caso que la pieza contenga un número real
    '@param numero Número de tipo dim  as double
    '@param nivel Nivel en que se encuentra en la expresión este número
    Public Sub New(ByVal numero As Double, ByVal nivel As Integer)
        Me.tipo = 1 'Es un número
        Me.numero = numero
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = numero
    End Sub

    'Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
    '@param operador Que puede ser +, -, *, /, ^
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal operador As Char, ByVal nivel As Integer)

        Me.tipo = 2 'Es un operador
        Me.operador = operador
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = 0
    End Sub


    'Constructor en caso que el nodo contenga una variable
    '@param variable Puede ir de 0 a 25 que representa de a..z
    '@param nivel Nivel en que se encuentra en la expresión este operador
    Public Sub New(ByVal variable As Integer, ByVal nivel As Integer)
        Me.tipo = 3 'Es una variable
        Me.variable = variable
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = 0
    End Sub


    'Constructor en caso que la pieza contenga una función
    '@param funcion Identificación de la función
    '@param nivel Nivel en que se encuentra en la expresión este operador
    '@param bandera Sólo sirve para diferenciarlo del anterior constructor
    Public Sub New(ByVal funcion As Integer, ByVal nivel As Integer, ByVal bandera As String)
        Me.tipo = 4 'Es una función
        Me.funcion = funcion
        Me.nivel = nivel

        Me.evaluado = False
        Me.acumula = 0
    End Sub

    'Retorna el acumulado que tiene esta pieza
    '@return Acumulado de la pieza
    Public Function getAcumula() As Double
```

```
        Return acumula
    End Function

    'Retorna si la pieza ya fue evaluada
    '@return True si la pieza ya fue evaluada
    Public Function isEvaluado() As Boolean
        Return evaluado
    End Function


    'Retorna el número de tipo dim  as double que tiene la pieza
    '@return El valor del número tipo dim  as double
    Public Function getNumero() As Double
        Return numero
    End Function


    'Retorna el operador (+, -, *, /, ^) que tiene la pieza
    '@return El operador en dim  as char
    Public Function getOperador() As Char
        Return operador
    End Function


    'Retorna la variable que tiene la pieza
    '@return La variable
    Public Function getVariable() As Integer
        Return variable
    End Function


    'Retorna que tipo de pieza es: número, operador, variable o función
    '@return Tipo de pieza
    Public Function getTipo() As Integer
        Return tipo
    End Function


    'Retorna en qué nivel se encuentra la pieza con respecto a la expresión
    '@return Nivel
    Public Function getNivel() As Integer
        Return nivel
    End Function


    'Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
    '@return Código de la función
    Public Function getFuncion() As Integer
        Return funcion
    End Function


    'Da valor al acumulado por pieza
    '@param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
    Public Sub setAcumula(ByVal acumulado As Double)
        Me.acumula = acumulado
    End Sub

    'Marca la pieza como ya evaluada
    '@param evaluado true si la pieza ya ha sido evaluada, false si no
    Public Sub setEvaluado(ByVal evaluado As Boolean)
        Me.evaluado = evaluado
    End Sub

End Class
```

EvaluadorExpresionAlgebraica.vb

```
'Autor: Rafael Alberto Moreno Parra
'Correo: ramsoftware@gmail.com
'Página Web: http://darwin.50webs.com
'Licencia: LGPL
'Fecha: Enero de 2012

Public Class EvaluadorExpresionAlgebraica
  'Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
  Private Shared ESNUMERO As Integer = 1
  Private Shared ESOPERADOR As Integer = 2
  Private Shared ESVARIABLE As Integer = 3
  Private Shared ESFUNCION As Integer = 4

  'Esta constante sirve para que se reste al caracter y se obtenga el número
  'Ejemplo:  '7' - ASCIINUMERO =  7
  Private Shared ASCIINUMERO As Integer = 48

  'Esta constante sirve para que se reste al caracter y se obtenga el número
  'usado en el arreglo unidimensional que alberga los valores de las variables
  'Ejemplo:  'b' - ASCIILETRA =  1
  Private Shared ASCIILETRA As Integer = 97

  'Las funciones que soporta este evaluador
```

```
Private Shared TAMANOFUNCION As Integer = 39
Private Shared listaFunciones As [String] = "sinsencostanabsasnacsatnlogceiexpsqrrcb"

'Lista simplemente enlazada que tiene los componentes (número, operador o función)
'de la expresión algebraica ya dividida.
Private Piezas As New List(Of Pieza)()

'Arreglo unidimensional con las 26 variables diferentes
Private valorVariable As Double() = New Double(25) {}

'Almacena hasta que nivel se llega en paréntesis internos
Private MaximoNivel As Integer

'Variable que se pone a true cuando hay un error matemático
Private ERRORMATEMATICO As Boolean

'Este método se encarga de analizar la expresión y convertirla en una
'estructura que permita evaluar la expresión.
'     *
'Convierte una expresión algebraica en una sucesión de nodos.
'|2| |+| |a| |/| |5| |*| |cos| |y|
'|2| |+| |a| |/| |5| |*| |cos| |y|
'
'@param expr La expresión algebraica sin espacios y en minúsculas
Public Sub Analizar(ByVal expr As [String])
  ' Inicializa la lista de piezas
  Piezas.Clear()

  ' Inicializa el nivel
Dim nivel As Integer = 0
MaximoNivel = 0

' Tamaño de la expresión simple
Dim longitud As Integer = expr.Length

' Va extrayendo letra a letra de la expresión simple
Dim letra As Char

' Conversión de string a double
Dim entero As Integer = 0, fraccion As Integer = 0, divide As Integer = 1
Dim puntoentero As Boolean = False

' Si llevaba acumulando un valor numérico esta variable se vuelve true
Dim acumulabaNumero As Boolean = False

For cont As Integer = 0 To longitud - 1
  ' Trae la letra
  letra = expr(cont)

  ' Si hay un paréntesis que abre, el nivel aumenta
  If letra = "(" Then
    nivel += 1
    If nivel > MaximoNivel Then
      MaximoNivel = nivel
    End If
    ' Si es paréntesis que cierra, el nivel disminuye
  ElseIf letra = ")" Then
    nivel -= 1
    ' Si es una variable o una función
  ElseIf letra >= "a" AndAlso letra <= "z" Then
    ' Detecta si es una función porque tiene dos letras seguidas
    If cont < longitud - 1 Then
      ' Chequea si el siguiente carácter es una letra, dado el caso es una función
      Dim letra2 As Char = expr(cont + 1)
      If letra2 >= "a" AndAlso letra2 <= "z" Then
        Dim letra3 As Char = expr(cont + 2)

        ' Identifica las funciones
        Dim funcionDetectada As Integer = 1
        For funcion As Integer = 0 To TAMANOFUNCION Step 3
          If letra = listaFunciones(funcion) AndAlso letra2 = listaFunciones(funcion + 1) AndAlso letra3 =
listaFunciones(funcion + 2) Then
            Exit For
          End If
          funcionDetectada += 1
        Next

        ' Adiciona a la lista
        Dim objeto As New Pieza(funcionDetectada, nivel, "f")
        Piezas.Add(objeto)

        nivel += 1
        If nivel > MaximoNivel Then
          MaximoNivel = nivel
        End If

        ' Mueve tres caracteres  sin(  [s][i][n][(]
        cont += 3
      Else
        ' Es una variable, no una función
        Dim objeto As New Pieza(Asc(letra) - ASCIILETRA, nivel)
        Piezas.Add(objeto)
```

```
        End If
      Else
        ' Es una variable, no una función
        Dim objeto As New Pieza(Asc(letra) - ASCIILETRA, nivel)
        Piezas.Add(objeto)
      End If
      ' Si es un número

    ElseIf (letra >= "0" AndAlso letra <= "9") OrElse letra = "." Then
      ' Ir armando el número de tipo double
      If letra = "." Then
        puntoentero = True
      ElseIf Not puntoentero Then
        ' puntoentero == false
        entero = entero * 10 + Asc(letra) - ASCIINUMERO
      Else
        fraccion = fraccion * 10 + Asc(letra) - ASCIINUMERO
        divide *= 10
      End If
      acumulabaNumero = True
    Else
      ' Si es un operador entonces crea el nodo del operador y el
      ' nodo del número si venía acumulando un número

      If acumulabaNumero Then
        ' El nodo del número
        Dim objeto As New Pieza(entero + CDbl(fraccion) / divide, nivel)

        ' Agrega a la lista
        Piezas.Add(objeto)
      End If

      ' El nodo operador
      Dim objeto2 As New Pieza(letra, nivel)

      ' Agrega a la lista
      Piezas.Add(objeto2)

      ' Inicializa de nuevo las variables de conversión de string a número
      entero = 0
      fraccion = 0
      divide = 1
      puntoentero = False
      acumulabaNumero = False
    End If
  Next

  ' Cierra la expresión simple preguntando si el último operando es un número
  If acumulabaNumero Then
    Dim objeto As New Pieza(entero + CDbl(fraccion) / divide, nivel)
    Piezas.Add(objeto)
  End If
End Sub

'Ya construida la lista del tipo:
'[nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
'es ir del operador con más precedencia al de menos precedencia.
'
'Este método se llama después de haber sido analizada la expresión.
'
'En el caso que sólo cambia el valor de una variable, no es necesario
'analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
'la misma expresión.
'
'@return El valor de la expresión evaluada (double)
Public Function Evaluar() As Double
  Dim pos As Integer = 0, antes As Integer = 0, sigue As Integer = 0
  ERRORMATEMATICO = False

  ' Total de nodos en la lista creada
  Dim totalPiezas As Integer = Piezas.Count

  For pos = 0 To totalPiezas - 1
    ' Activa todas las piezas para que sean evaluadas
    Piezas(pos).setEvaluado(False)

    ' Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
    ' ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables.
    If Piezas(pos).getTipo() = ESVARIABLE Then
      Piezas(pos).setAcumula(valorVariable(Piezas(pos).getVariable()))
    ElseIf Piezas(pos).getTipo() = ESNUMERO Then
      Piezas(pos).setAcumula(Piezas(pos).getNumero())
    End If
  Next

  ' Va del nivel mas profundo al mas superficial
  For evaluaNivel As Integer = MaximoNivel To 0 Step -1
    ' Recorre toda la lista
    For pos = 0 To totalPiezas - 1
      ' Si encuentra una pieza de tipo función la evalúa con el valor de la siguiente pieza
      If Piezas(pos).getNivel() = evaluaNivel AndAlso Piezas(pos).getTipo() = ESFUNCION Then
        Select Case Piezas(pos).getFuncion()
```

```
            Case 1, 2
              Piezas(pos).setAcumula(Math.Sin(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 3
              Piezas(pos).setAcumula(Math.Cos(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 4
              Piezas(pos).setAcumula(Math.Tan(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 5
              If Piezas(pos + 1).getAcumula() > 0 Then
                Piezas(pos).setAcumula(Piezas(pos + 1).getAcumula())
              Else
                Piezas(pos).setAcumula(-Piezas(pos + 1).getAcumula())
              End If
              Exit Select
            Case 6
              If Piezas(pos + 1).getAcumula() >= -1 AndAlso Piezas(pos + 1).getAcumula() <= 1 Then
                Piezas(pos).setAcumula(Math.Asin(Piezas(pos + 1).getAcumula()))
              Else
                ERRORMATEMATICO = True
                Return 0
              End If
              Exit Select
            Case 7
              If Piezas(pos + 1).getAcumula() >= -1 AndAlso Piezas(pos + 1).getAcumula() <= 1 Then
                Piezas(pos).setAcumula(Math.Acos(Piezas(pos + 1).getAcumula()))
              Else
                ERRORMATEMATICO = True
                Return 0
              End If
              Exit Select
            Case 8
              Piezas(pos).setAcumula(Math.Atan(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 9
              Piezas(pos).setAcumula(Math.Log(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 10
              Piezas(pos).setAcumula(Math.Ceiling(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 11
              Piezas(pos).setAcumula(Math.Exp(Piezas(pos + 1).getAcumula()))
              Exit Select
            Case 12
              If Piezas(pos + 1).getAcumula() >= 0 Then
                Piezas(pos).setAcumula(Math.Sqrt(Piezas(pos + 1).getAcumula()))
              Else
                ERRORMATEMATICO = True
                Return 0
              End If
              Exit Select
            Case 13
              Piezas(pos).setAcumula(Math.Pow(Piezas(pos + 1).getAcumula(), CDbl(1) / 3))
              Exit Select
          End Select

          ' Marca el nodo siguiente como ya evaluado
          Piezas(pos + 1).setEvaluado(True)
        End If
      Next

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es exponente
        If Piezas(pos).getNivel() = evaluaNivel AndAlso Piezas(pos).getTipo() = ESOPERADOR AndAlso Piezas(pos).getOperador() =
"^" Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While

          ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
          sigue = pos + 1
          While Piezas(sigue).isEvaluado()
            sigue += 1
          End While

          ' Marca esos nodos actual y siguiente como ya evaluados
          Piezas(pos).setEvaluado(True)
          Piezas(sigue).setEvaluado(True)

          ' Hace la operación de Número elevado a Número
          Piezas(antes).setAcumula(Math.Pow(Piezas(antes).getAcumula(), Piezas(sigue).getAcumula()))
        End If
      Next

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es multiplicación o división
```

```
        If Piezas(pos).getNivel() = evaluaNivel AndAlso Piezas(pos).getTipo() = ESOPERADOR AndAlso (Piezas(pos).getOperador() =
"*" OrElse Piezas(pos).getOperador() = "/") Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While

          ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
          sigue = pos + 1
          While Piezas(sigue).isEvaluado()
            sigue += 1
          End While

          ' Marca esos nodos actual y siguiente como ya evaluados
          Piezas(pos).setEvaluado(True)
          Piezas(sigue).setEvaluado(True)

          ' Hace la operación de Número * Número, o Número / Número
          If Piezas(pos).getOperador() = "*" Then
            Piezas(antes).setAcumula(Piezas(antes).getAcumula() * Piezas(sigue).getAcumula())
          Else
            If Piezas(sigue).getAcumula() <> 0 Then
              Piezas(antes).setAcumula(Piezas(antes).getAcumula() / Piezas(sigue).getAcumula())
            Else
              ERRORMATEMATICO = True
              Return 0
            End If

          End If
        End If
      Next

      ' Recorre toda la lista
      For pos = 0 To totalPiezas - 1
        ' Si encuentra un nodo del tipo operador y es suma o resta
        If Piezas(pos).getNivel() = evaluaNivel AndAlso Piezas(pos).getTipo() = ESOPERADOR AndAlso (Piezas(pos).getOperador() =
"+" OrElse Piezas(pos).getOperador() = "-") Then
          ' Busca un nodo anterior que sea número o variable y no haya sido evaluado
          antes = pos - 1
          While Piezas(antes).isEvaluado()
            antes -= 1
          End While

          ' Busca un nodo siguiente que sea número o variable y no haya sido evaluado
          sigue = pos + 1
          While Piezas(sigue).isEvaluado()
            sigue += 1
          End While

          ' Marca esos nodos actual y siguiente como ya evaluados
          Piezas(pos).setEvaluado(True)
          Piezas(sigue).setEvaluado(True)

          ' Hace la operación de Número + Número, o Número - Número
          If Piezas(pos).getOperador() = "+" Then
            Piezas(antes).setAcumula(Piezas(antes).getAcumula() + Piezas(sigue).getAcumula())
          Else
            Piezas(antes).setAcumula(Piezas(antes).getAcumula() - Piezas(sigue).getAcumula())
          End If
        End If
      Next
    Next

    ' Resultado de la evaluación
    Return Piezas(0).getAcumula()
  End Function

  'Guarda en un arreglo unidimensional el valor de las variables
  '@param variable ¿Qué variable de la expresión recibirá el valor? a..z
  '@param valor  Valor de tipo double que tendrá esa variable.
  Public Sub DarValorVariable(ByVal variable As Char, ByVal valor As Double)
    valorVariable(Asc(variable) - ASCIILETRA) = valor
  End Sub

  'Retorna true si hubo un error matemático
  '@return TRUE si hubo un error matemático
  Public Function getErrorMatematico() As Boolean
    Return ERRORMATEMATICO
  End Function

End Class
```

Module1.vb

```
Module Module1

    Sub Main()
        Dim objEvaluador As New EvaluadorExpresionAlgebraica
        Dim valor As Double
        Dim expresion As String
```

```
        expresion = "abs(7.325)-2.180+tan(2.476)-9.834/x"
        objEvaluador.Analizar(expresion)
        objEvaluador.DarValorVariable("x", 13.1729)
        valor = objEvaluador.Evaluar()
        Console.WriteLine(valor)
        Console.ReadKey()
    End Sub

End Module
```

Java

Pieza.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
public class Pieza
{
    /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    private int nivel;

    /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
    private int tipo;

    /* El número en la expresión algebraica */
    private double numero;

    /* El operador (+, -, *, /, ^) en la expresión algebraica */
    private char operador;

    /* La variable */
    private int variable;

    /* La función */
    private int funcion;

    /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
       cálculos que determinan el valor de la expresión */
    private boolean evaluado;

    /* El resultado parcial de la evaluación de la expresión */
    private double acumula;

    /**
     * Constructor en caso que la pieza contenga un número real
     * @param numero Número de tipo double
     * @param nivel Nivel en que se encuentra en la expresión este número
     */
    public Pieza(double numero, int nivel)
    {
     this.tipo = 1; //Es un número
     this.numero = numero;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = numero;
    }

    /**
     * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
     * @param operador Que puede ser +, -, *, /, ^
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(char operador, int nivel)
    {
     this.tipo = 2; //Es un operador
     this.operador = operador;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = 0;
    }

    /**
     * Constructor en caso que el nodo contenga una variable
     * @param variable Puede ir de 0 a 25 que representa de a..z
     * @param nivel Nivel en que se encuentra en la expresión este operador
     */
    public Pieza(int variable, int nivel)
    {
     this.tipo = 3; //Es una variable
     this.variable = variable;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = 0;
    }

    /**
     * Constructor en caso que la pieza contenga una función
     * @param funcion Identificación de la función
     * @param nivel Nivel en que se encuentra en la expresión este operador
     * @param bandera Sólo sirve para diferenciarlo del anterior constructor
     */
    public Pieza(int funcion, int nivel, char bandera)
    {
```

```
     this.tipo = 4; //Es una función
     this.funcion = funcion;
     this.nivel = nivel;

     this.evaluado = false;
     this.acumula = 0;
    }

    /** Retorna el acumulado que tiene esta pieza
     * @return Acumulado de la pieza
     */
    public double getAcumula()
    {
     return acumula;
    }

    /** Retorna si la pieza ya fue evaluada
     * @return True si la pieza ya fue evaluada
     */
    public boolean isEvaluado()
    {
     return evaluado;
    }

    /**
     * Retorna el número de tipo double que tiene la pieza
     * @return El valor del número tipo double
     */
    public double getNumero()
    {
     return numero;
    }

    /**
     * Retorna el operador (+, -, *, /, ^) que tiene la pieza
     * @return El operador en char
     */
    public char getOperador()
    {
     return operador;
    }

    /**
     * Retorna la variable que tiene la pieza
     * @return La variable
     */
    public int getVariable()
    {
     return variable;
    }

    /**
     * Retorna que tipo de pieza es: número, operador, variable o función
     * @return Tipo de pieza
     */
    public int getTipo()
    {
     return tipo;
    }

    /**
     * Retorna en que nivel se encuentra la pieza con respecto a la expresión
     * @return Nivel
     */
    public int getNivel()
    {
            return nivel;
    }

    /**
     * Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
     * @return Código de la función
     */
    public int getFuncion()
    {
            return funcion;
    }

    /**
     * Da valor al acumulado por pieza
     * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
     */
    public void setAcumula(double acumulado)
    {
     this.acumula = acumulado;
    }

    /**
     * Marca la pieza como ya evaluada
     * @param evaluado true si la pieza ya ha sido evaluada, false si no
     */
    void setEvaluado(boolean evaluado)
```

```
    {
     this.evaluado = evaluado;
    }
}
```

EvaluadorExpresionAlgebraica.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
import java.util.ArrayList;

public class EvaluadorExpresionAlgebraica
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  private static final int ESNUMERO = 1;
  private static final int ESOPERADOR = 2;
  private static final int ESVARIABLE = 3;
  private static final int ESFUNCION = 4;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * Ejemplo:  '7' - ASCIINUMERO =  7
   */
  private static final int ASCIINUMERO = 48;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * usado en el arreglo unidimensional que alberga los valores de las variables
   * Ejemplo:  'b' - ASCIILETRA =  1
   */
  private static final int ASCIILETRA = 97;

  /**
   * Las funciones que soporta este evaluador
   */
  private static final int TAMANOFUNCION = 39;
  private static final String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

  /**
   * Lista simplemente enlazada que tiene los componentes (número, operador o función)
   * de la expresión algebraica ya dividida.
   */
  private ArrayList<Pieza> Piezas = new ArrayList<Pieza>();

  /**
   * Arreglo unidimensional con las 26 variables diferentes
   */
  private double valorVariable[] = new double[26];

  /**
   * Almacena hasta que nivel se llega en paréntesis internos
   */
  private int MaximoNivel;

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  private boolean ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |a| |/| |5| |*| |cos| |y|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  public void Analizar(String expr)
  {
    /* Inicializa la lista de piezas  */
    Piezas.clear();

    /* Inicializa el nivel */
    int nivel = 0;
    MaximoNivel = 0;

    /* Tamaño de la expresión simple */
    int longitud = expr.length();

    /* Va extrayendo letra a letra de la expresión simple */
    char letra;
```

```
/* Conversión de string a double */
int decimal = 0, fraccion = 0, divide = 1;
boolean puntodecimal = false;

/* Si llevaba acumulando un valor numérico esta variable se vuelve true */
boolean acumulabaNumero = false;

for (int cont=0; cont<longitud; cont++)
{
  /* Trae la letra */
  letra = expr.charAt(cont);

  /* Si hay un paréntesis que abre, el nivel aumenta */
  if (letra=='(')
  {
    nivel++;
    if (nivel>MaximoNivel) MaximoNivel = nivel;
  }
  /* Si es paréntesis que cierra, el nivel disminuye */
  else if (letra==')')
    nivel--;
  /* Si es una variable o una función */
  else if (letra >= 'a' && letra <= 'z')
  {
    /* Detecta si es una función porque tiene dos letras seguidas */
    if (cont < longitud-1)
    {
      /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
      char letra2 = expr.charAt(cont+1);
      if ( letra2 >= 'a' && letra2 <= 'z')
      {
        char letra3 = expr.charAt(cont+2);

        /* Identifica las funciones */
        int funcionDetectada = 1;
        for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
        {
          if (letra == listaFunciones.charAt(funcion)
            && letra2 == listaFunciones.charAt(funcion + 1)
            && letra3 == listaFunciones.charAt(funcion + 2))
          break;
          funcionDetectada++;
        }

        /* Adiciona a la lista */
        Pieza objeto = new Pieza(funcionDetectada, nivel, 'f');
        Piezas.add(objeto);

        nivel++;
        if (nivel>MaximoNivel) MaximoNivel = nivel;

        /* Mueve tres caracteres  sin(  [s][i][n][(] */
        cont+=3;
      }
      /* Es una variable, no una función */
      else
      {
        Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
        Piezas.add(objeto);
      }
    }
    /* Es una variable, no una función */
    else
    {
      Pieza objeto = new Pieza(letra - ASCIILETRA, nivel);
      Piezas.add(objeto);
    }
  }
  /* Si es un número */
  else if( (letra >= '0' && letra <= '9') || letra == '.')
  {
    /* Ir armando el número de tipo double */
    if (letra == '.')
      puntodecimal = true;
    else
      if(!puntodecimal)  /* puntodecimal == false */
        decimal = decimal * 10 + letra-ASCIINUMERO;
      else
      {
        fraccion = fraccion * 10 + letra-ASCIINUMERO;
        divide *=10;
      }
      acumulabaNumero = true;
  }
  else
  {
    /* Si es un operador entonces crea el nodo del operador y el
      nodo del número si venía acumulando un número */
    if (acumulabaNumero)
    {
      /* El nodo del número */
```

```
          Pieza objeto = new Pieza(decimal + (double) fraccion/divide, nivel);

          /* Agrega a la lista */
          Piezas.add(objeto);
        }

        /* El nodo operador */
        Pieza objeto = new Pieza(letra, nivel);

        /* Agrega a la lista */
        Piezas.add(objeto);

        /* Inicializa de nuevo las variables de conversión de string a número */
        decimal = fraccion = 0;
        divide = 1;
        puntodecimal = acumulabaNumero = false;
      }
    }

    /* Cierra la expresión simple preguntando si el último operando es un número */
    if (acumulabaNumero)
    {
      Pieza objeto = new Pieza(decimal + (double) fraccion/divide, nivel);
      Piezas.add(objeto);
    }
  }

  /**
   * Ya construida la lista del tipo:
   * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   * es ir del operador con mas precedencia al de menos precedencia.
   *
   * Este método se llama después de haber sido analizada la expresión.
   *
   * En el caso que sólo cambia el valor de una variable, no es necesario
   * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
   * la misma expresión.
   *
   * @return El valor de la expresión evaluada (double)
   */
  public double Evaluar()
  {
    int pos=0, antes=0, sigue=0;
    ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    int totalPiezas = Piezas.size();

    for (pos=0; pos<totalPiezas; pos++)
    {
      /* Activa todas las piezas para que sean evaluadas */
      Piezas.get(pos).setEvaluado(false);

      /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
         ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
      if (Piezas.get(pos).getTipo() == ESVARIABLE)
        Piezas.get(pos).setAcumula(valorVariable[Piezas.get(pos).getVariable()]);
      else if (Piezas.get(pos).getTipo() == ESNUMERO)
        Piezas.get(pos).setAcumula(Piezas.get(pos).getNumero());
    }

    /* Va del nivel mas profundo al mas superficial */
    for (int evaluaNivel = MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
    {
      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra una pieza de tipo función la evalúa con el valor de la siguiente pieza */
        if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESFUNCION )
        {
          switch (Piezas.get(pos).getFuncion())
          {
            case 1:
            case 2:
              Piezas.get(pos).setAcumula(Math.sin(Piezas.get(pos+1).getAcumula()));
              break;
            case 3:
              Piezas.get(pos).setAcumula(Math.cos(Piezas.get(pos+1).getAcumula()));
              break;
            case 4:
              Piezas.get(pos).setAcumula(Math.tan(Piezas.get(pos+1).getAcumula()));
              break;
            case 5:
              if (Piezas.get(pos+1).getAcumula() > 0)
                Piezas.get(pos).setAcumula(Piezas.get(pos+1).getAcumula());
              else
                Piezas.get(pos).setAcumula(-Piezas.get(pos+1).getAcumula());
              break;
            case 6:
              if (Piezas.get(pos+1).getAcumula() >= -1 && Piezas.get(pos+1).getAcumula() <= 1)
                Piezas.get(pos).setAcumula(Math.asin(Piezas.get(pos+1).getAcumula()));
              else
```

```
          {
            ERRORMATEMATICO = true;
            return 0;
          }
          break;
        case 7:
          if (Piezas.get(pos+1).getAcumula() >= -1 && Piezas.get(pos+1).getAcumula() <= 1)
            Piezas.get(pos).setAcumula(Math.acos(Piezas.get(pos+1).getAcumula()));
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
          break;
        case 8:
          Piezas.get(pos).setAcumula(Math.atan(Piezas.get(pos+1).getAcumula()));
          break;
        case 9:
          Piezas.get(pos).setAcumula(Math.log(Piezas.get(pos+1).getAcumula()));
          break;
        case 10:
          Piezas.get(pos).setAcumula(Math.ceil(Piezas.get(pos+1).getAcumula()));
          break;
        case 11:
          Piezas.get(pos).setAcumula(Math.exp(Piezas.get(pos+1).getAcumula()));
          break;
        case 12:
          if (Piezas.get(pos+1).getAcumula() >= 0)
            Piezas.get(pos).setAcumula(Math.sqrt(Piezas.get(pos+1).getAcumula()));
          else
          {
            ERRORMATEMATICO = true;
            return 0;
          }
          break;
        case 13:
          Piezas.get(pos).setAcumula(Math.pow(Piezas.get(pos+1).getAcumula(), (double) 1 / 3));
          break;
      }

      /* Marca el nodo siguiente como ya evaluado */
      Piezas.get(pos+1).setEvaluado(true);
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es exponente */
    if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
Piezas.get(pos).getOperador() == '^')
    {
      /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
      for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
      for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas.get(pos).setEvaluado(true);
      Piezas.get(sigue).setEvaluado(true);

      /* Hace la operación de Número elevado a Número */
      Piezas.get(antes).setAcumula(Math.pow(Piezas.get(antes).getAcumula(), Piezas.get(sigue).getAcumula()));
    }
  }

  /* Recorre toda la lista */
  for (pos=0; pos<totalPiezas; pos++)
  {
    /* Si encuentra un nodo del tipo operador y es multiplicación o división */
    if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
(Piezas.get(pos).getOperador() == '*' || Piezas.get(pos).getOperador() == '/'))
    {
      /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
      for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

      /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
      for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

      /* Marca esos nodos actual y siguiente como ya evaluados */
      Piezas.get(pos).setEvaluado(true);
      Piezas.get(sigue).setEvaluado(true);

      /* Hace la operación de Número * Número, o Número / Número */
      if (Piezas.get(pos).getOperador() == '*')
        Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() *  Piezas.get(sigue).getAcumula() );
      else
      {
        if (Piezas.get(sigue).getAcumula() != 0)
          Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() /  Piezas.get(sigue).getAcumula() );
        else
```

```
            {
              ERRORMATEMATICO = true;
              return 0;
            }
          }

        }
      }

      /* Recorre toda la lista */
      for (pos=0; pos<totalPiezas; pos++)
      {
        /* Si encuentra un nodo del tipo operador y es suma o resta */
        if (Piezas.get(pos).getNivel() == evaluaNivel && Piezas.get(pos).getTipo() == ESOPERADOR &&
(Piezas.get(pos).getOperador() == '+' || Piezas.get(pos).getOperador() == '-'))
        {
          /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
          for (antes=pos-1; Piezas.get(antes).isEvaluado(); antes--);

          /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
          for (sigue=pos+1; Piezas.get(sigue).isEvaluado(); sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          Piezas.get(pos).setEvaluado(true);
          Piezas.get(sigue).setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if (Piezas.get(pos).getOperador() == '+')
            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() +  Piezas.get(sigue).getAcumula() );
          else
            Piezas.get(antes).setAcumula( Piezas.get(antes).getAcumula() -  Piezas.get(sigue).getAcumula() );
        }
      }
    }

    /* Resultado de la expresión */
    return Piezas.get(0).getAcumula();
  }

  /**
   * Guarda en un arreglo unidimensional el valor de las variables
   * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
   * @param valor  Valor de tipo double que tendrá esa variable.
   */
  public void DarValorVariable(char variable, double valor)
  {
     valorVariable[variable - ASCIILETRA] = valor;
  }

  /**
   * Retorna true si hubo un error matemático
   * @return TRUE si hubo un error matemático
   */
  public boolean getErrorMatematico()
  {
      return ERRORMATEMATICO;
  }
}
```

Inicio.java

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
public class Inicio
{
    public static void main(String args[])
    {
        EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
        String expresion;
        double valor;
        expresion = "tan(cos(6.305+cos(1.371*1.669)*x))";
        objEvaluador.Analizar(expresion);
        objEvaluador.DarValorVariable('x', 13.1729);
        valor = objEvaluador.Evaluar();
        System.out.println(valor);
    }
}
```

PHP

Pieza.php

```php
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
class Pieza
{
  /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
  var $nivel;

  /* 1. Almacena un número, 2. Almacena un operador, 3. Almacena una variable, 4. Almacena una función */
  var $tipo;

  /* El número en la expresión algebraica */
  var $numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  var $operador;

  /* La variable */
  var $variable;

  /* La función */
  var $funcion;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  var $evaluado;

  /* El resultado parcial de la evaluación de la expresión */
  var $acumula;

  /**
   * Constructor en caso que la pieza contenga un número real
   * @param numero Número de tipo double
   */
  public function ConstructorNumero($numero, $nivel)
  {
   $this->tipo = 1; //Es un número
   $this->numero = $numero;
   $this->nivel = $nivel;

   $this->evaluado = false;
   $this->acumula = $numero;
  }

  /**
   * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
   * @param operador Que puede ser +, -, *, /, ^
   */
  public function ConstructorOperador($operador, $nivel)
  {
    $this->tipo = 2; //Es un operador
    $this->operador = $operador;
    $this->nivel = $nivel;

    $this->evaluado = false;
    $this->acumula = 0;
  }

  /**
   * Constructor en caso que el nodo contenga una variable
   * @param variable Puede ir de 0 a 25 que representa de a..z
   * @param nivel Nivel en que se encuentra en la expresión este operador
   */
  public function ConstructorVariable($variable, $nivel)
  {
    $this->tipo = 3; //Es una variable
    $this->variable = $variable;
    $this->nivel = $nivel;

    $this->evaluado = false;
    $this->acumula = 0;
  }

  /**
   * Constructor en caso que la pieza contenga una función
   * @param funcion Identificación de la función
   * @param nivel Nivel en que se encuentra en la expresión este operador
   */
  public function ConstructorFuncion($funcion, $nivel)
  {
    $this->tipo = 4; //Es una función
    $this->funcion = $funcion;
```

```
    $this->nivel = $nivel;

    $this->evaluado = false;
    $this->acumula = 0;
  }

  /** Retorna el acumulado que tiene esta pieza
   * @return Acumulado de la pieza
   */
  public function getAcumula()
  {
    return $this->acumula;
  }

  /** Retorna si la pieza ya fue evaluada
   * @return True si la pieza ya fue evaluada
   */
  function isEvaluado()
  {
    return $this->evaluado;
  }

  /**
   * Retorna el número de tipo double que tiene la pieza
   * @return El valor del número tipo double
   */
  public function getNumero()
  {
    return $this->numero;
  }

  /**
   * Retorna el operador (+, -, *, /, ^) que tiene la pieza
   * @return El operador en char
   */
  public function getOperador()
  {
    return $this->operador;
  }

  /**
   * Retorna la variable que tiene la pieza
   * @return La variable
   */
  public function getVariable()
  {
   return $this->variable;
  }

  /**
   * Retorna que tipo de pieza es: número u operador
   * @return Tipo de pieza
   */
  public function getTipo()
  {
    return $this->tipo;
  }

  /**
   * Retorna en que nivel se encuentra la pieza con respecto a la expresión
   * @return Nivel
   */
  public function getNivel()
  {
     return $this->nivel;
  }

  /**
   * Retorna el código de la función (abs, sen, cos, tan) que tiene la piez
   * @return Código de la función
   */
  public function getFuncion()
  {
     return $this->funcion;
  }

  /**
   * Da valor al acumulado por pieza
   * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
   */
  public function setAcumula($acumulado)
  {
    $this->acumula = $acumulado;
  }

  /**
   * Marca la pieza como ya evaluada
   * @param evaluado true si la pieza ya ha sido evaluada, false si no
   */
  public function setEvaluado($evaluado)
  {
    $this->evaluado = $evaluado;
```

```
  }
}
?>
```

EvaluadorExpresionAlgebraica.php

```
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
include("Pieza.php");

class EvaluadorExpresionAlgebraica
{
  /**
   * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
   */
  var $ESNUMERO = 1;
  var $ESOPERADOR = 2;
  var $ESVARIABLE = 3;
  var $ESFUNCION = 4;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * Ejemplo:  '7' - ASCIINUMERO =  7
   */
  var $ASCIINUMERO = 48;

  /**
   * Esta constante sirve para que se reste al carácter y se obtenga el número
   * usado en el arreglo unidimensional que alberga los valores de las variables
   * Ejemplo:  'b' - ASCIILETRA =  1
   */
  var $ASCIILETRA = 97;

   /**
   * Las funciones que soporta este evaluador
   */
  var $TAMANOFUNCION = 39;
  var $listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

  /**
   * Lista simplemente enlazada que tiene los componentes (número u operador)
   * de la expresión algebraica ya dividida.
   */
  var $Piezas = array();

  /**
   * Arreglo unidimensional con las 26 variables diferentes
   */
  var $valorVariable = array();

  /**
   * Almacena hasta que nivel se llega en paréntesis internos
   */
  var $MaximoNivel;

  /**
   * Variable que se pone a true cuando hay un error matemático
   */
  var $ERRORMATEMATICO;

  /**
   * Este método se encarga de analizar la expresión y convertirla en una
   * estructura que permita evaluar la expresión.
   *
   * Convierte una expresión algebraica en una sucesión de nodos.
   * |2| |+| |8.3| |/| |5| |*| |7.4|
   *
   * @param expr La expresión algebraica sin espacios y en minúsculas
  */
  public function Analizar($expr)
  {
    /* Inicializa la lista de piezas  */
    unset($this->Piezas);

    /* Inicializa el nivel */
    $nivel = 0;
    $this->MaximoNivel = 0;

    /* Tamaño de la expresión simple */
    $longitud = strlen($expr);

    /* Conversión de string a double */
    $entero = 0;
    $fraccion = 0;
    $divide = 1;
```

```
    $puntoentero = false;

    /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
    $acumulabaNumero = false;

    for ($cont=0; $cont<$longitud; $cont++)
    {
      /* Trae la letra */
      $letra = $expr{$cont};

      /* Si hay un paréntesis que abre, el nivel aumenta */
      if ($letra=='(')
      {
        $nivel++;
        if ($nivel>$this->MaximoNivel) $this->MaximoNivel = $nivel;
      }
      /* Si es paréntesis que cierra, el nivel disminuye */
      else if ($letra==')')
        $nivel--;
      /* Si es una variable o una función */
      else if ($letra >= 'a' && $letra <= 'z')
      {
        /* Detecta si es una función porque tiene dos letras seguidas */
        if ($cont < $longitud-1)
        {
          /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
          $letra2 = $expr{$cont+1};
          if ( $letra2 >= 'a' && $letra2 <= 'z')
          {
            $letra3 = $expr{$cont+2};

            /* Identifica las funciones */
            $funcionDetectada = 1;
            for ($funcion = 0; $funcion <= $this->TAMANOFUNCION; $funcion += 3)
            {
              if ($letra == $this->listaFunciones{$funcion}
                && $letra2 == $this->listaFunciones{$funcion + 1}
                && $letra3 == $this->listaFunciones{$funcion + 2})
                break;
              $funcionDetectada++;
            }

            /* Adiciona a la lista */
            $objeto = new Pieza();
            $objeto->ConstructorFuncion($funcionDetectada, $nivel);
            $this->Piezas[] = $objeto;

            $nivel++;
            if ($nivel > $this->MaximoNivel) $this->MaximoNivel = $nivel;

            /* Mueve tres caracteres  sin(  [s][i][n][(] */
            $cont+=3;
          }
          /* Es una variable, no una función */
          else
          {
            $objeto = new Pieza();
            $objeto->ConstructorVariable(ord($letra) - $this->ASCIILETRA, $nivel);
            $this->Piezas[] = $objeto;
          }
        }
        /* Es una variable, no una función */
        else
        {
          $objeto = new Pieza();
          $objeto->ConstructorVariable(ord($letra) - $this->ASCIILETRA, $nivel);
          $this->Piezas[] = $objeto;
        }
      }
      /* Si es un número */
      else if( ($letra >= '0' && $letra <= '9') || $letra == '.')
      {
        /* Ir armando el número de tipo double */
        if ($letra == '.')
          $puntoentero = true;
        else
          if(!$puntoentero)  /* $puntoentero == false */
            $entero = $entero * 10 + $letra;
          else
          {
            $fraccion = $fraccion * 10 + $letra;
            $divide *=10;
          }
          $acumulabaNumero = true;
      }
      else
      {
        /* Si es un operador entonces crea el nodo del operador y el
        nodo del número si venía acumulando un número */
        if ($acumulabaNumero)
        {
          $objeto = new Pieza();
```

```
          $objeto->ConstructorNumero($entero + $fraccion/$divide, $nivel);
          $this->Piezas[] = $objeto;
        }

        /* Agrega a la lista */
        $objeto = new Pieza();
        $objeto->ConstructorOperador($letra, $nivel);
        $this->Piezas[] = $objeto;

        /* Inicializa de nuevo las variables de conversión de string a número */
        $entero = 0;
        $fraccion = 0;
        $divide = 1;
        $puntoentero = false;
        $acumulabaNumero = false;
      }
    }

    /* Cierra la expresión simple preguntando si el último operando es un número */
    if ($acumulabaNumero)
    {
      $objeto = new Pieza();
      $objeto->ConstructorNumero($entero + $fraccion/$divide, $nivel);
      $this->Piezas[] = $objeto;
    }
  }

  /**
   * Ya construida la lista del tipo:
   * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
   * es ir del operador con mas precedencia al de menos precedencia.
   *
   * Este método se llama después de haber sido analizada la expresión.
   *
   * @return El valor de la expresión evaluada (double)
   */
  public function Evaluar()
  {
    $pos=0;
    $antes=0;
    $sigue=0;
    $this->ERRORMATEMATICO = false;

    /* Total de nodos en la lista creada */
    $totalPiezas = sizeof($this->Piezas);

    for ($pos=0; $pos<$totalPiezas; $pos++)
    {
       /* Activa todas las piezas para que sean evaluadas */
       $this->Piezas[$pos]->setEvaluado(false);

       /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
          ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
       if ($this->Piezas[$pos]->getTipo() == $this->ESVARIABLE)
         $this->Piezas[$pos]->setAcumula($this->valorVariable[$this->Piezas[$pos]->getVariable()]);
       else if ($this->Piezas[$pos]->getTipo() == $this->ESNUMERO)
         $this->Piezas[$pos]->setAcumula($this->Piezas[$pos]->getNumero());
    }

    /* Va del nivel mas profundo al mas superficial */
    for ($evaluaNivel = $this->MaximoNivel; $evaluaNivel >= 0; $evaluaNivel--)
    {
      /* Recorre toda la lista */
      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
         /* Si encuentra una pieza de tipo función la evalúa con el valor de la siguiente pieza */
         if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESFUNCION )
         {
           switch ($this->Piezas[$pos]->getFuncion())
           {
             case 1:
             case 2:
                $this->Piezas[$pos]->setAcumula(sin($this->Piezas[$pos+1]->getAcumula()));
                break;
             case 3:
                $this->Piezas[$pos]->setAcumula(cos($this->Piezas[$pos+1]->getAcumula()));
                break;
             case 4:
                $this->Piezas[$pos]->setAcumula(tan($this->Piezas[$pos+1]->getAcumula()));
                break;
             case 5:
                if ($this->Piezas[$pos+1]->getAcumula() > 0)
                  $this->Piezas[$pos]->setAcumula($this->Piezas[$pos+1]->getAcumula());
                else
                  $this->Piezas[$pos]->setAcumula(-$this->Piezas[$pos+1]->getAcumula());
                break;
             case 6:
                if ($this->Piezas[$pos+1]->getAcumula() >= -1 && $this->Piezas[$pos+1]->getAcumula() <= 1)
                   $this->Piezas[$pos]->setAcumula(asin($this->Piezas[$pos+1]->getAcumula()));
                else
                {
                  $this->ERRORMATEMATICO = true;
```

```
            return 0;
         }
         break;
      case 7:
         if ($this->Piezas[$pos+1]->getAcumula() >= -1 && $this->Piezas[$pos+1]->getAcumula() <= 1)
            $this->Piezas[$pos]->setAcumula(acos($this->Piezas[$pos+1]->getAcumula()));
         else
         {
            $this->ERRORMATEMATICO = true;
            return 0;
         }
         break;
      case 8:
         $this->Piezas[$pos]->setAcumula(atan($this->Piezas[$pos+1]->getAcumula()));
         break;
      case 9:
         $this->Piezas[$pos]->setAcumula(log($this->Piezas[$pos+1]->getAcumula()));
         break;
      case 10:
         $this->Piezas[$pos]->setAcumula(ceil($this->Piezas[$pos+1]->getAcumula()));
         break;
      case 11:
         $this->Piezas[$pos]->setAcumula(exp($this->Piezas[$pos+1]->getAcumula()));
         break;
      case 12:
         if ($this->Piezas[$pos+1]->getAcumula() >= 0)
            $this->Piezas[$pos]->setAcumula(sqrt($this->Piezas[$pos+1]->getAcumula()));
         else
         {
            $this->ERRORMATEMATICO = true;
            return 0;
         }
         break;
      case 13:
         $this->Piezas[$pos]->setAcumula(pow($this->Piezas[$pos+1]->getAcumula(), (double) 1 / 3));
         break;
      }

      /* Marca el nodo siguiente como ya evaluado */
      $this->Piezas[$pos+1]->setEvaluado(true);
     }
   }


   /* Recorre toda la lista */
   for ($pos=0; $pos<$totalPiezas; $pos++)
   {
     /* Si encuentra un nodo del tipo operador y es exponente */
     if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && $this->Piezas[$pos]->getOperador() == '^')
     {
       /* Busca un nodo anterior que sea número y no haya sido evaluado */
       for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

       /* Busca un nodo siguiente que sea número y no haya sido evaluado */
       for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

       /* Marca esos nodos actual y siguiente como ya evaluados */
       $this->Piezas[$pos]->setEvaluado(true);
       $this->Piezas[$sigue]->setEvaluado(true);

       /* Hace la operación de Número elevado a Número */
       $this->Piezas[$antes]->setAcumula(pow($this->Piezas[$antes]->getAcumula(), $this->Piezas[$sigue]->getAcumula()));
     }
   }

   /* Recorre toda la lista */
   for ($pos=0; $pos<$totalPiezas; $pos++)
   {
     /* Si encuentra un nodo del tipo operador y es multiplicación o división */
     if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this->Piezas[$pos]->getOperador() == '*' || $this->Piezas[$pos]->getOperador() == '/'))
     {
       /* Busca un nodo anterior que sea número y no haya sido evaluado */
       for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

       /* Busca un nodo siguiente que sea número y no haya sido evaluado */
       for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

       /* Marca esos nodos actual y siguiente como ya evaluados */
       $this->Piezas[$pos]->setEvaluado(true);
       $this->Piezas[$sigue]->setEvaluado(true);

       /* Hace la operación de Número * Número, o Número / Número */
       if ($this->Piezas[$pos]->getOperador() == '*')
          $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() * $this->Piezas[$sigue]->getAcumula() );
       else
       {
          if ($this->Piezas[$sigue]->getAcumula() != 0)
             $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() / $this->Piezas[$sigue]->getAcumula() );
          else
          {
```

```
                $this->ERRORMATEMATICO = true;
                return 0;
            }
          }
        }
      }

      /* Recorre toda la lista */
      for ($pos=0; $pos<$totalPiezas; $pos++)
      {
        /* Si encuentra un nodo del tipo operador y es suma o resta */
        if ($this->Piezas[$pos]->getNivel() == $evaluaNivel && $this->Piezas[$pos]->getTipo() == $this->ESOPERADOR && ($this-
>Piezas[$pos]->getOperador() == '+' || $this->Piezas[$pos]->getOperador() == '-'))
        {
          /* Busca un nodo anterior que sea número y no haya sido evaluado */
          for ($antes=$pos-1; $this->Piezas[$antes]->isEvaluado(); $antes--);

          /* Busca un nodo siguiente que sea número y no haya sido evaluado */
          for ($sigue=$pos+1; $this->Piezas[$sigue]->isEvaluado(); $sigue++);

          /* Marca esos nodos actual y siguiente como ya evaluados */
          $this->Piezas[$pos]->setEvaluado(true);
          $this->Piezas[$sigue]->setEvaluado(true);

          /* Hace la operación de Número + Número, o Número - Número */
          if ($this->Piezas[$pos]->getOperador() == '+')
             $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() + $this->Piezas[$sigue]->getAcumula() );
          else
             $this->Piezas[$antes]->setAcumula( $this->Piezas[$antes]->getAcumula() - $this->Piezas[$sigue]->getAcumula() );
        }
      }
    }
    /* Resultado de la expresión */
    return $this->Piezas[0]->getAcumula();
  }

  /**
  * Guarda en un arreglo unidimensional el valor de las variables
  * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
  * @param valor Valor de tipo double que tendrá esa variable.
  */
  public function DarValorVariable($variable, $valor)
  {
  $this->valorVariable[ord($variable) - $this->ASCIILETRA] = $valor;
  }

   /**
    * Retorna true si hubo un error matemático
    * @return TRUE si hubo un error matemático
    */
   public function getERRORMATEMATICO()
   {
       return $this->ERRORMATEMATICO;
   }
}
?>
```

index.php

```
<?php
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
include("EvaluadorExpresionAlgebraica.php");

$objEvaluador = new EvaluadorExpresionAlgebraica();


$expresion = '4.697-cos(0.481)+cos(8.110/(abs(x)))';
$objEvaluador->Analizar($expresion);
$objEvaluador->DarValorVariable('x', 13.1729);
$valor = $objEvaluador->Evaluar();
echo $valor;
?>
```

JavaScript

EvaluadorExpresionAlgebraica.js

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
function Pieza()
{
  /* Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
  this.nivel;

  /* 1. Almacena un número, 2. Almacena un operador */
  this.tipo;

  /* El número en la expresión algebraica */
  this.numero;

  /* El operador (+, -, *, /, ^) en la expresión algebraica */
  this.operador;

  /* Determina si esta pieza ya ha sido evaluada en el momento de hacer los
     cálculos que determinan el valor de la expresión */
  this.evaluado;

/* El resultado parcial de la evaluación de la expresión */
this.acumula;

/**
 * Constructor en caso que la pieza contenga un número real
 * @param numero Número de tipo double
 */
this.ConstructorNumero = function(numero)
{
 this.tipo = 1; //Es un número
 this.numero = numero;

 this.evaluado = false;
 this.acumula = numero;
}

/**
 * Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
 * @param operador Que puede ser +, -, *, /, ^
 */
this.ConstructorOperador = function(operador, nivel)
{
  this.tipo = 2; //Es un operador
  this.operador = operador;
  this.nivel = nivel;

  this.evaluado = false;
  this.acumula = 0;
}

/**
 * Constructor en caso que el nodo contenga una variable
 * @param variable Puede ir de 0 a 25 que representa de a..z
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
this.ConstructorVariable = function(variable, nivel)
{
 this.tipo = 3; //Es una variable
 this.variable = variable;
 this.nivel = nivel;

 this.evaluado = false;
 this.acumula = 0;
}

/**
 * Constructor en caso que la pieza contenga una función
 * @param funcion Identificación de la función
 * @param nivel Nivel en que se encuentra en la expresión este operador
 */
this.ConstructorFuncion = function(funcion, nivel)
{
 this.tipo = 4; //Es una función
 this.funcion = funcion;
 this.nivel = nivel;

 this.evaluado = false;
 this.acumula = 0;
}

/** Retorna el acumulado que tiene esta pieza
 * @return Acumulado de la pieza
```

```
 */
this.getAcumula = function()
{
  return this.acumula;
}

/** Retorna si la pieza ya fue evaluada
 * @return True si la pieza ya fue evaluada
 */
this.isEvaluado = function()
{
  return this.evaluado;
}

/**
 * Retorna el número de tipo double que tiene la pieza
 * @return El valor del número tipo double
 */
this.getNumero = function()
{
  return this.numero;
}

/**
 * Retorna el operador (+, -, *, /, ^) que tiene la pieza
 * @return El operador en char
 */
this.getOperador = function()
{
  return this.operador;
}

/**
 * Retorna la variable que tiene la pieza
 * @return La variable
 */
this.getVariable = function()
{
  return this.variable;
}

/**
 * Retorna que tipo de pieza es: número u operador
 * @return Tipo de pieza
 */
this.getTipo = function()
{
  return this.tipo;
}

/**
 * Retorna en que nivel se encuentra la pieza con respecto a la expresión
 * @return Nivel
 */
this.getNivel = function()
{
   return this.nivel;
}

/**
 * Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
 * @return Código de la función
 */
this.getFuncion = function()
{
    return this.funcion;
}

/**
 * Da valor al acumulado por pieza
 * @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
 */
this.setAcumula = function(acumulado)
{
  this.acumula = acumulado;
}

/**
 * Marca la pieza como ya evaluada
 * @param evaluado true si la pieza ya ha sido evaluada, false si no
 */
this.setEvaluado = function(evaluado)
{
  this.evaluado = evaluado;
}
}

/**
  ************************************************************************************************************


  ************************************************************************************************************
```

```
*/


function EvaluadorExpresionAlgebraica()
/**
 * Evaluador de expresiones algebraicas.
 * Recibe la expresión algebraica por ejemplo: "sin(x)+cos(y)*3.14"
 * como una cadena (String) junto con el valor de las variables (x=8.92, y=-7.43)
 * el algoritmo interpreta esa expresión respetando las reglas
 * algebraicas y retorna el resultado de tipo double.
 *
 * @author Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web:  http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha:  Mayo de 2011
 */
{
    /**
     * Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
     */
    this.ESNUMERO = 1;
    this.ESOPERADOR = 2;
    this.ESVARIABLE = 3;
    this.ESFUNCION = 4;

    /**
     * Esta constante sirve para que se reste al carácter y se obtenga el número
     * usado en el arreglo unidimensional que alberga los valores de las variables
     * Ejemplo:  'b' - ASCIILETRA =  1
     */
    this.ASCIILETRA = 97;

    /**
     * Las funciones que soporta este evaluador
     */
    this.TAMANOFUNCION = 39;
    this.listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

    /**
     * Lista simplemente enlazada que tiene los componentes (número, operador o función)
     * de la expresión algebraica ya dividida.
     */
    this.Piezas = [];

    /**
     * Arreglo unidimensional con las 26 variables diferentes
     */
    this.valorVariable = [];

    /**
     * Almacena hasta que nivel se llega en paréntesis internos
     */
    this.MaximoNivel;

    /**
     * Variable que se pone a true cuando hay un error matemático
     */
    this.ERRORMATEMATICO;

    /**
     * Este método se encarga de analizar la expresión y convertirla en una
     * estructura que permita evaluar la expresión.
     *
     * Convierte una expresión algebraica en una sucesión de nodos.
     * |2| |+| |a| |/| |5| |*| |cos| |y|
     *
     * @param expr La expresión algebraica sin espacios y en minúsculas
    */
    this.Analizar = function(expr)
    {
        /* Inicializa la lista de piezas  */
        this.Piezas.length = 0;
        NumeroPieza = 0;

        /* Inicializa el nivel */
        nivel = 0;
        this.MaximoNivel = 0;

        /* Tamaño de la expresión simple */
        longitud = expr.length;

        /* Conversión de string a double */
        entero = 0;
        fraccion = 0;
        divide = 1;
        puntoentero = false;

        /* Si llevaba acumulando un valor numérico esta variable se vuelve true */
        acumulabaNumero = false;

        for (cont=0; cont<longitud; cont++)
```

```
        {
            /* Trae la letra */
            letra = expr.charAt(cont);

            /* Si hay un paréntesis que abre, el nivel aumenta */
            if (letra=='(')
            {
                nivel++;
                if (nivel>this.MaximoNivel) this.MaximoNivel = nivel;
            }
            /* Si es paréntesis que cierra, el nivel disminuye */
            else if (letra==')')
                nivel--;
            /* Si es una variable o una función */
            else if (letra >= 'a' && letra <= 'z')
            {
                /* Detecta si es una función porque tiene dos letras seguidas */
                if (cont < longitud-1)
                {
                    /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
                    letra2 = expr.charAt(cont+1);
                    if ( letra2 >= 'a' && letra2 <= 'z')
                    {
                        letra3 = expr.charAt(cont+2);

                        /* Identifica las funciones */
                        funcionDetectada = 1;
                        for (funcion = 0; funcion <= this.TAMANOFUNCION; funcion += 3)
                        {
                            if (letra == this.listaFunciones.charAt(funcion)
                                && letra2 == this.listaFunciones.charAt(funcion + 1)
                                && letra3 == this.listaFunciones.charAt(funcion + 2))
                            break;
                            funcionDetectada++;
                        }

                        /* Adiciona a la lista */
                        objeto = new Pieza();
                        objeto.ConstructorFuncion(funcionDetectada, nivel);
                        this.Piezas[NumeroPieza++] = objeto;

                        nivel++;
                        if (nivel>this.MaximoNivel) this.MaximoNivel = nivel;

                        /* Mueve tres caracteres  sin(  [s][i][n][(] */
                        cont+=3;
                    }
                    /* Es una variable, no una función */
                    else
                    {
                        objeto = new Pieza();
                        objeto.ConstructorVariable(letra - this.ASCIILETRA, nivel);
                        this.Piezas[NumeroPieza++] = objeto;
                    }
                }
                /* Es una variable, no una función */
                else
                {
                    objeto = new Pieza();
                    objeto.ConstructorVariable(letra - this.ASCIILETRA, nivel);
                    this.Piezas[NumeroPieza++] = objeto;
                }
            }
            /* Si es un número */
            else if( (letra >= '0' && letra <= '9') || letra == '.')
            {
                /* Ir armando el número de tipo double */
                if (letra == '.')
                    puntoentero = true;
                else
                    if(!puntoentero)  /* puntoentero == false */
                        entero = entero * 10 + parseFloat(letra);
                    else
                    {
                        fraccion = fraccion * 10 + parseFloat(letra);
                        divide *=10;
                    }
                    acumulabaNumero = true;
            }
            else
            {
                /* Si es un operador entonces crea el nodo del operador y el
                    nodo del número si venía acumulando un número */
                if (acumulabaNumero)
                {
                    /* El nodo del número */
                    objeto = new Pieza();
                    objeto.ConstructorNumero(entero + fraccion/divide, nivel);
                    this.Piezas[NumeroPieza++] = objeto;
                }

                /* El nodo operador */
```

```
                objeto = new Pieza();
                objeto.ConstructorOperador(letra, nivel);

                /* Agrega a la lista */
                this.Piezas[NumeroPieza++] = objeto;

                /* Inicializa de nuevo las variables de conversión de string a número */
                entero = fraccion = 0;
                divide = 1;
                puntoentero = acumulabaNumero = false;
            }
        }

        /* Cierra la expresión simple preguntando si el último operando es un número */
        if (acumulabaNumero)
        {
            objeto = new Pieza();
            objeto.ConstructorNumero(entero + fraccion/divide, nivel);
            this.Piezas[NumeroPieza++] = objeto;
        }
    }

    /**
     * Ya construida la lista del tipo:
     * [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
     * es ir del operador con mas precedencia al de menos precedencia.
     *
     * Este método se llama después de haber sido analizada la expresión.
     *
     * En el caso que sólo cambia el valor de una variable, no es necesario
     * analizar de nuevo la expresión, luego es muy rápido evaluar múltiples veces
     * la misma expresión.
     *
     * @return El valor de la expresión evaluada (double)
     */
    this.Evaluar = function()
    {
        pos=0, antes=0, sigue=0;
        ERRORMATEMATICO = false;

        /* Total de nodos en la lista creada */
        totalPiezas = this.Piezas.length;

        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Activa todas las piezas para que sean evaluadas */
            this.Piezas[pos].setEvaluado(false);

            /* Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
               ¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
            if (this.Piezas[pos].getTipo() == this.ESVARIABLE)
                this.Piezas[pos].setAcumula(this.valorVariable[this.Piezas[pos].getVariable()]);
            else if (this.Piezas[pos].getTipo() == this.ESNUMERO)
                this.Piezas[pos].setAcumula(this.Piezas[pos].getNumero());
        }

        /* Va del nivel mas profundo al mas superficial */
        for (evaluaNivel = this.MaximoNivel; evaluaNivel >= 0; evaluaNivel--)
        {
            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra una pieza de tipo función la evalúa con el valor de la siguiente pieza */
                if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESFUNCION )
                {
                    switch (this.Piezas[pos].getFuncion())
                    {
                        case 1:
                        case 2:
                            this.Piezas[pos].setAcumula(Math.sin(this.Piezas[pos+1].getAcumula()));
                            break;
                        case 3:
                            this.Piezas[pos].setAcumula(Math.cos(this.Piezas[pos+1].getAcumula()));
                            break;
                        case 4:
                            this.Piezas[pos].setAcumula(Math.tan(this.Piezas[pos+1].getAcumula()));
                            break;
                        case 5:
                            if (this.Piezas[pos+1].getAcumula() > 0)
                                this.Piezas[pos].setAcumula(this.Piezas[pos+1].getAcumula());
                            else
                                this.Piezas[pos].setAcumula(-this.Piezas[pos+1].getAcumula());
                            break;
                        case 6:
                            if (this.Piezas[pos+1].getAcumula() >= -1 && this.Piezas[pos+1].getAcumula() <= 1)
                                this.Piezas[pos].setAcumula(Math.asin(this.Piezas[pos+1].getAcumula()));
                            else
                            {
                                ERRORMATEMATICO = true;
                                return 0;
                            }
                            break;
```

```
                case 7:
                    if (this.Piezas[pos+1].getAcumula() >= -1 && this.Piezas[pos+1].getAcumula() <= 1)
                        this.Piezas[pos].setAcumula(Math.acos(this.Piezas[pos+1].getAcumula()));
                    else
                    {
                        ERRORMATEMATICO = true;
                        return 0;
                    }
                    break;
                case 8:
                    this.Piezas[pos].setAcumula(Math.atan(this.Piezas[pos+1].getAcumula()));
                    break;
                case 9:
                    this.Piezas[pos].setAcumula(Math.log(this.Piezas[pos+1].getAcumula()));
                    break;
                case 10:
                    this.Piezas[pos].setAcumula(Math.ceil(this.Piezas[pos+1].getAcumula()));
                    break;
                case 11:
                    this.Piezas[pos].setAcumula(Math.exp(this.Piezas[pos+1].getAcumula()));
                    break;
                case 12:
                    if (this.Piezas[pos+1].getAcumula() >= 0)
                        this.Piezas[pos].setAcumula(Math.sqrt(this.Piezas[pos+1].getAcumula()));
                    else
                    {
                        ERRORMATEMATICO = true;
                        return 0;
                    }
                    break;
                case 13:
                    this.Piezas[pos].setAcumula(Math.pow(this.Piezas[pos+1].getAcumula(), 1 / 3));
                    break;
                }

                /* Marca el nodo siguiente como ya evaluado */
                this.Piezas[pos+1].setEvaluado(true);
            }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es exponente */
            if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
this.Piezas[pos].getOperador() == '^')
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                this.Piezas[pos].setEvaluado(true);
                this.Piezas[sigue].setEvaluado(true);

                /* Hace la operación de Número elevado a Número */
                this.Piezas[antes].setAcumula(Math.pow(this.Piezas[antes].getAcumula(), this.Piezas[sigue].getAcumula()));
            }
        }

        /* Recorre toda la lista */
        for (pos=0; pos<totalPiezas; pos++)
        {
            /* Si encuentra un nodo del tipo operador y es multiplicación o división */
            if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
(this.Piezas[pos].getOperador() == '*' || this.Piezas[pos].getOperador() == '/'))
            {
                /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                /* Marca esos nodos actual y siguiente como ya evaluados */
                this.Piezas[pos].setEvaluado(true);
                this.Piezas[sigue].setEvaluado(true);

                /* Hace la operación de Número * Número, o Número / Número */
                if (this.Piezas[pos].getOperador() == '*')
                    this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() *  this.Piezas[sigue].getAcumula() );
                else
                {
                    if (this.Piezas[sigue].getAcumula() != 0)
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() /  this.Piezas[sigue].getAcumula() );
                    else
                    {
                        ERRORMATEMATICO = true;
                        return 0;
                    }
                }
```

```
                }
            }

            /* Recorre toda la lista */
            for (pos=0; pos<totalPiezas; pos++)
            {
                /* Si encuentra un nodo del tipo operador y es suma o resta */
                if (this.Piezas[pos].getNivel() == evaluaNivel && this.Piezas[pos].getTipo() == this.ESOPERADOR &&
(this.Piezas[pos].getOperador() == '+' || this.Piezas[pos].getOperador() == '-'))
                {
                    /* Busca un nodo anterior que sea número o variable y no haya sido evaluado */
                    for (antes=pos-1; this.Piezas[antes].isEvaluado(); antes--);

                    /* Busca un nodo siguiente que sea número o variable y no haya sido evaluado */
                    for (sigue=pos+1; this.Piezas[sigue].isEvaluado(); sigue++);

                    /* Marca esos nodos actual y siguiente como ya evaluados */
                    this.Piezas[pos].setEvaluado(true);
                    this.Piezas[sigue].setEvaluado(true);

                    /* Hace la operación de Número + Número, o Número - Número */
                    if (this.Piezas[pos].getOperador() == '+')
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() +  this.Piezas[sigue].getAcumula() );
                    else
                        this.Piezas[antes].setAcumula( this.Piezas[antes].getAcumula() -  this.Piezas[sigue].getAcumula() );
                }
            }
        }

        /* Resultado de la expresión */
        return this.Piezas[0].getAcumula();
    }

    /**
     * Guarda en un arreglo unidimensional el valor de las variables
     * @param variable ¿Qué variable de la expresión recibirá el valor? a..z
     * @param valor  Valor de tipo double que tendrá esa variable.
     */
    this.DarValorVariable = function(variable, valor)
    {
         this.valorVariable[variable - this.ASCIILETRA] = valor;
    }

    /**
     * Retorna true si hubo un error matemático
     * @return TRUE si hubo un error matemático
     */
    this.getErrorMatematico = function()
    {
            return ERRORMATEMATICO;
    }
}
```

index.html

```
<html>
  <body>
    <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
      <script type="text/javascript">
      /* Probando el evaluador de expresiones */
      objEvaluador = new EvaluadorExpresionAlgebraica();


      expresion = "x+3.517-tan((2.887/x)*cos(x))";
      objEvaluador.Analizar(expresion);
      objEvaluador.DarValorVariable('x', 13.1729);
      valor = objEvaluador.Evaluar();
      document.write(valor);
      </script>
   </body>
</html>
```

Object Pascal

Pieza.pas

```
//
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web: http://darwin.50webs.com
// Licencia: LGPL
// Fecha: Enero de 2012
//
unit Pieza;

interface
type
  TPieza = class
  private
    // Nivel en la expresión, a mayor valor, es mayor profundidad en la expresión */
    nivel: integer;

    // 1. Almacena un número, 2. Almacena un operador
    tipo: integer;

    // El número en la expresión algebraica */
    numero: double;

    // El operador (+, -, *, /, ^) en la expresión algebraica
    operador: char;

    // La variable
    variable: integer;

    // La función
    funcion: integer;

    //Determina si esta pieza ya ha sido evaluada en el momento de hacer los
    //cálculos que determinan el valor de la expresión
    evaluado: boolean;

    //El resultado parcial de la evaluación de la expresión */
    acumula: double;

   public
     Constructor Create(numero: double; nivel: integer); overload;
     Constructor Create(operador: char; nivel: integer); overload;
     Constructor Create(variable: integer; nivel: integer); overload;
     Constructor Create(funcion: integer; nivel: integer; bandera: char); overload;
     function getAcumula(): double;
     function isEvaluado(): boolean;
     function getNumero(): double;
     function getOperador(): char;
     function getVariable(): integer;
     function getTipo(): integer;
     function getNivel(): integer;
     function getFuncion(): integer;
     procedure setAcumula(acumulado: double);
     procedure setEvaluado(evaluado: boolean);
   end;
implementation


// Constructor en caso que la pieza contenga un número real
// @param numero Número de tipo double
// @param nivel Nivel en que se encuentra en la expresión este operador
constructor TPieza.Create(numero: double; nivel: integer);
begin
   self.tipo := 1; //Es un número
   self.numero := numero;
   self.nivel := nivel;

   self.evaluado := false;
   self.acumula := numero;
end;


// Constructor en caso que el nodo contenga un operador (+, -, *, /, ^)
// @param operador Que puede ser +, -, *, /, ^
// @param nivel Nivel en que se encuentra en la expresión este operador
constructor TPieza.Create(operador: char; nivel: integer);
begin
    self.tipo := 2; //Es un operador
    self.operador := operador;
    self.nivel := nivel;

    self.evaluado := false;
    self.acumula := 0;
end;


// Constructor en caso que el nodo contenga una variable
// @param variable Puede ir de 0 a 25 que representa de a..z
```

```
// @param nivel Nivel en que se encuentra en la expresión esta variable
constructor TPieza.Create(variable: integer; nivel: integer);
begin
     self.tipo := 3; //Es una variable
     self.variable := variable;
     self.nivel := nivel;

     self.evaluado := false;
     self.acumula := 0;
end;

//Constructor en caso que la pieza contenga una función
//@param funcion Identificación de la función
//@param nivel Nivel en que se encuentra en la expresión este operador
//@param bandera Sólo sirve para diferenciarlo del anterior constructor
constructor TPieza.Create(funcion: integer; nivel: integer; bandera: char);
begin
     self.tipo := 4; //Es una función
     self.funcion := funcion;
     self.nivel := nivel;

     self.evaluado := false;
     self.acumula := 0;
end;

// Retorna el acumulado que tiene esta pieza
// @return Acumulado de la pieza
function TPieza.getAcumula(): double;
begin
    getAcumula := acumula;
end;

// Retorna si la pieza ya fue evaluada
// @return True si la pieza ya fue evaluada
function TPieza.isEvaluado(): boolean;
begin
    isEvaluado := evaluado;
end;


// Retorna el número de tipo double que tiene la pieza
// @return El valor del número tipo double
function TPieza.getNumero(): double;
begin
    getNumero := numero;
end;

// Retorna el operador (+, -, *, /, ^) que tiene la pieza
// @return El operador en char
function TPieza.getOperador(): char;
begin
    getOperador := operador;
end;

// Retorna la variable que tiene la pieza
// @return La variable
function TPieza.getVariable(): integer;
begin
    getVariable := variable;
end;

// Retorna que tipo de pieza es: número u operador
// @return Tipo de pieza
function TPieza.getTipo(): integer;
begin
    getTipo := tipo;
end;

// Retorna en que nivel se encuentra la pieza con respecto a la expresión
// @return Nivel
function TPieza.getNivel(): integer;
begin
    getNivel := nivel;
end;

// Retorna el código de la función (abs, sen, cos, tan) que tiene la pieza
// @return Código de la función
function TPieza.getFuncion(): integer;
begin
    getFuncion := funcion;
end;

// Da valor al acumulado por pieza
// @param acumulado  Acumulado que nace de cada operación simple es dado a la pieza aquí.
procedure TPieza.setAcumula(acumulado:double);
begin
    self.acumula := acumulado;
end;

// Marca la pieza como ya evaluada
// @param evaluado true si la pieza ya ha sido evaluada, false si no
procedure TPieza.setEvaluado(evaluado: boolean);
```

```
begin
    self.evaluado := evaluado;
end;

end.
```

EvaluadorExpresionAlgebraica.pas

```
// Autor: Rafael Alberto Moreno Parra
// Correo: ramsoftware@gmail.com
// Página Web:  http://darwin.50webs.com
// Licencia: LGPL
// Fecha:  Marzo de 2012
unit EvaluadorExpresionAlgebraica;

interface
uses
  //Requerido para el TobjectList
  Contnrs, SysUtils, Math, Pieza;

type
  TEvaluadorExpresionAlgebraica = class
  private
      //Constantes para determinar que tipo es cada pieza en que se divide la expresión algebraica
      ESNUMERO: integer;
      ESOPERADOR: integer;
      ESVARIABLE: integer;
      ESFUNCION: integer;

      //Esta constante sirve para que se reste al carácter y se obtenga el número
      //Ejemplo:  '7' - ASCIINUMERO =  7
      ASCIINUMERO: integer;

      //Esta constante sirve para que se reste al carácter y se obtenga el número
      //usado en el arreglo unidimensional que alberga los valores de las variables
      //Ejemplo:  'b' - ASCIILETRA =  1
      ASCIILETRA: integer;

      // Las funciones que soporta este evaluador
      TAMANOFUNCION: integer;
      listaFunciones: string;

      //Lista simplemente enlazada que tiene los componentes (número u operador)
      //de la expresión algebraica ya dividida.
      Piezas: TobjectList;
      objPieza: TPieza;

      //Arreglo unidimensional con las 26 variables diferentes
      valorVariable: array[0..26] of double;

      //Almacena hasta que nivel se llega en paréntesis internos
      MaximoNivel: integer;

      //Variable que se pone a true cuando hay un error matemático
      ERRORMATEMATICO: boolean;
  public
      Constructor Create();
      procedure Analizar(expr: string);
      function Evaluar(): double;
      function getErrorMatematico(): boolean;
      procedure DarValorVariable(variable: char; valor: double);
      procedure LiberaMemoria();
  end;

implementation

constructor TEvaluadorExpresionAlgebraica.Create();
begin
    self.ESNUMERO := 1;
    self.ESOPERADOR := 2;
    self.ESVARIABLE := 3;
    self.ESFUNCION := 4;
    self.ASCIINUMERO := 48;
    self.ASCIILETRA := 97;
    self.TAMANOFUNCION := 39;
    self.listaFunciones := 'sinsencostanabsasnacsatnlogceiexpsqrrcb';
end;

//Este método se encarga de analizar la expresión y convertirla en una
//estructura que permita evaluar la expresión.
//
//Convierte una expresión algebraica en una sucesión de nodos.
//|2| |+| |8.3| |/| |5| |*| |7.4|
//
//@param expr La expresión algebraica sin espacios y en minúsculas

procedure TEvaluadorExpresionAlgebraica.Analizar(expr: string);
var
  longitud: integer;

  //Va extrayendo letra a letra de la expresión simple */
  letra: char;
```

```
  //Conversión de string a double */
  entero, fraccion, divide: integer;
  puntoentero: boolean;

  //Si llevaba acumulando un valor numérico esta variable se vuelve true */
  acumulabaNumero: boolean;

  cont: integer;

  //Nivel
  nivel: integer;

  //Funciones
  letra2, letra3: char;
  funcion, funciondetectada: integer;

begin
  // Inicializa la lista de piezas
  Piezas := TObjectList.Create;

  // Tamaño de la expresión simple
  longitud := length(expr);

  // Conversión de string a double
  entero := 0;
  fraccion := 0;
  divide := 1;
  puntoentero := false;

// Si llevaba acumulando un valor numérico esta variable se vuelve true */
acumulabaNumero := false;

//Inicializa el nivel
nivel := 0;
MaximoNivel := 0;

cont := 1;
while cont <= longitud do
begin
  //Trae la letra
  letra := expr[cont];

  if letra='(' then
  begin
    nivel := nivel + 1;
    if nivel > MaximoNivel then MaximoNivel := nivel;
  end
  else if letra = ')' then
    nivel := nivel - 1
  // Si es una variable
  else if letra in ['a'..'z'] then
  begin
    if cont < longitud then
    begin
      letra2 := expr[cont+1];
      if letra2 in ['a'..'z'] then
      begin
        letra3 := expr[cont+2];

        funciondetectada := 1;
        funcion := 1;
        while (funcion <= TAMANOFUNCION) do
        begin
          if (letra = listaFunciones[funcion]) and (letra2 = listafunciones[funcion+1]) and (letra3 =
listafunciones[funcion+2]) then break;
          funcion := funcion + 3;
          funcionDetectada := funcionDetectada + 1;
        end;

        // Adiciona a la lista
        objPieza := TPieza.Create(funcionDetectada, nivel, 'f');
        Piezas.Add(objPieza);

        nivel := nivel + 1;
        if (nivel>MaximoNivel) then MaximoNivel := nivel;

        // Mueve tres caracteres  sin(  [s][i][n][(]
        cont := cont+3;
      end
      else
      // Es una variable, no una función
      begin
        objPieza := TPieza.Create(ord(letra) - ASCIILETRA, nivel);
        Piezas.Add(objPieza)
      end
    end
    else
    // Es una variable, no una función
    begin
      objPieza := TPieza.Create(ord(letra) - ASCIILETRA, nivel);
      Piezas.Add(objPieza);
```

```
    end
  end
  // Si es un número
  else if (letra in ['0'..'9']) or (letra = '.') then
  begin
    //Ir armando el número de tipo double */
    if letra = '.' then
      puntoentero := true
    else
    begin
      if not(puntoentero) then // puntoentero = false
        entero := entero * 10 + ord(letra) - ASCIINUMERO
      else
      begin
        fraccion := fraccion * 10 + ord(letra) - ASCIINUMERO;
        divide := divide*10;
      end;
      acumulabaNumero := true;
    end
  end
  else
  begin
    // Si es un operador entonces crea el nodo del operador y el
    // nodo del número si venía acumulando un número */
    if acumulabaNumero then
    begin
      objPieza := TPieza.Create(entero + fraccion/divide, nivel);
      Piezas.Add(objPieza);
    end;

      // Agrega a la lista el operador */
      objPieza := TPieza.Create(letra, nivel);
      Piezas.Add(objPieza);

      // Inicializa de nuevo las variables de conversión de string a número */
      entero := 0;
      fraccion := 0;
      divide := 1;
      puntoentero := false;
      acumulabaNumero := false;
    end;
    cont := cont + 1;
  end;

  // Cierra la expresión simple preguntando si el último operando es un número */
  if acumulabaNumero then
  begin
    objPieza := TPieza.Create(entero + fraccion/divide, nivel);
    Piezas.Add(objPieza);
  end;
end;

// Ya construida la lista del tipo:
// [nodo número]  [nodo operador]  [nodo número]  [nodo operador] .....
// es ir del operador con mas precedencia al de menos precedencia.
//
// Este método se llama después de haber sido analizada la expresión.
//
// @return El valor de la expresión evaluada (double)
function TEvaluadorExpresionAlgebraica.Evaluar(): double;
var
  pos, antes, sigue, totalPiezas: integer;
  evaluanivel: integer;
begin
  ERRORMATEMATICO := false;
try
  // Total de nodos en la lista creada
  totalPiezas := Piezas.Count - 1;

  for pos := 0 to totalPiezas do
  begin
    //Activa todas las piezas para que sean evaluadas */
    (Piezas.Items[pos] as TPieza).setEvaluado(false);

    // Recorre toda la lista poniendo los valores de las variables en el acumulado de cada pieza.
    //¿Cómo? Extrae el valor del arreglo unidimensional que alberga los valores de las variables. */
    if (Piezas.Items[pos] as Tpieza).getTipo() = ESVARIABLE then
       (Piezas.Items[pos] as Tpieza).setAcumula(valorVariable[(Piezas.Items[pos] as Tpieza).getVariable()])
    else if (Piezas.Items[pos] as Tpieza).getTipo() = ESNUMERO then
       (Piezas.Items[pos] as Tpieza).setAcumula((Piezas.Items[pos] as Tpieza).getNumero());
  end;

  // Va del nivel mas profundo al mas superficial */
  for evaluaNivel := MaximoNivel downto 0 do
  begin
    // Recorre toda la lista
    for pos:=0 to totalPiezas do
      if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() = ESFUNCION) then
      begin
        case (Piezas.Items[pos] as Tpieza).getFuncion() of
          1, 2: begin (Piezas.Items[pos] as Tpieza).setAcumula(sin((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          3: begin (Piezas.Items[pos] as Tpieza).setAcumula(cos((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
```

```
          4: begin (Piezas.Items[pos] as Tpieza).setAcumula(tan((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          5: begin (Piezas.Items[pos] as Tpieza).setAcumula(abs((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          6: begin (Piezas.Items[pos] as Tpieza).setAcumula(arcsin((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          7: begin (Piezas.Items[pos] as Tpieza).setAcumula(arccos((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          8: begin (Piezas.Items[pos] as Tpieza).setAcumula(arctan((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          9: begin (Piezas.Items[pos] as Tpieza).setAcumula(ln((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          10:begin (Piezas.Items[pos] as Tpieza).setAcumula(ceil((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          11:begin (Piezas.Items[pos] as Tpieza).setAcumula(exp((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          12:begin (Piezas.Items[pos] as Tpieza).setAcumula(sqrt((Piezas.Items[pos+1] as Tpieza).getAcumula())); end;
          13:begin (Piezas.Items[pos] as Tpieza).setAcumula(Power((Piezas.Items[pos+1] as Tpieza).getAcumula(), 1/3)); end;
        end;

        // Marca el nodo siguiente como ya evaluado */
        (Piezas.Items[pos+1] as Tpieza).setEvaluado(true);
      end;

        // Recorre toda la lista
        for pos :=0 to totalPiezas do
             //Si encuentra un nodo del tipo operador y es exponente */
             if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and ((Piezas.Items[pos] as Tpieza).getOperador() = '^') then
             begin
               //Busca un nodo anterior que sea número y no haya sido evaluado */
               antes := pos-1;
               while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

               //Busca un nodo siguiente que sea número y no haya sido evaluado */
               sigue := pos + 1;
               while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

               // Marca esos nodos actual y siguiente como ya evaluados */
               (Piezas.Items[pos] as Tpieza).setEvaluado(true);
               (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

               // Hace la operación de Número elevado a Número
               (Piezas.Items[antes] as Tpieza).setAcumula(Power((Piezas.Items[antes] as Tpieza).getAcumula(),
(Piezas.Items[sigue] as Tpieza).getAcumula()));
             end;

        // Recorre toda la lista
        for pos :=0 to totalPiezas do
             // Si encuentra un nodo del tipo operador y es multiplicación o división */
             if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '*') or ((Piezas.Items[pos] as Tpieza).getOperador() = '/'))
then
             begin
               //Busca un nodo anterior que sea número y no haya sido evaluado */
               antes := pos-1;
               while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

               //Busca un nodo siguiente que sea número y no haya sido evaluado */
               sigue := pos + 1;
               while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

               // Marca esos nodos actual y siguiente como ya evaluados */
               (Piezas.Items[pos] as Tpieza).setEvaluado(true);
               (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

               // Hace la operación de Número * Número, o Número / Número */
               if (Piezas.Items[pos] as Tpieza).getOperador() = '*' then
                    (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() *
(Piezas.Items[sigue] as Tpieza).getAcumula() )
               else
        begin
                    if (Piezas.Items[sigue] as Tpieza).getAcumula() <> 0 then
                      (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() /
(Piezas.Items[sigue] as Tpieza).getAcumula() )
                   else
                        begin
                          ERRORMATEMATICO := true;
                          Evaluar := 0;
                          Exit;
                        end;
               end;
             end;

        // Recorre toda la lista
        for pos :=0 to totalPiezas do
             // Si encuentra un nodo del tipo operador y es suma o resta */
             if ((Piezas.Items[pos] as Tpieza).getNivel() = evaluaNivel) and ((Piezas.Items[pos] as Tpieza).getTipo() =
ESOPERADOR) and (((Piezas.Items[pos] as Tpieza).getOperador() = '+') or ((Piezas.Items[pos] as Tpieza).getOperador() = '-'))
then
             begin
               //Busca un nodo anterior que sea número y no haya sido evaluado */
               antes := pos-1;
               while ((Piezas.Items[antes] as Tpieza).isEvaluado()) do antes := antes - 1;

               //Busca un nodo siguiente que sea número y no haya sido evaluado */
               sigue := pos + 1;
               while ((Piezas.Items[sigue] as Tpieza).isEvaluado()) do sigue := sigue + 1;

               // Marca esos nodos actual y siguiente como ya evaluados */
```

```
                (Piezas.Items[pos] as Tpieza).setEvaluado(true);
                (Piezas.Items[sigue] as Tpieza).setEvaluado(true);

                // Hace la operación de Número + Número, o Número - Número */
                if (Piezas.Items[pos] as Tpieza).getOperador() = '+' then
                      (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() +
(Piezas.Items[sigue] as Tpieza).getAcumula() )
                else
                      (Piezas.Items[antes] as Tpieza).setAcumula( (Piezas.Items[antes] as Tpieza).getAcumula() -
(Piezas.Items[sigue] as Tpieza).getAcumula() );
             end;
  end;

  // Resultado de la expresión
  Evaluar := (Piezas.Items[0] as Tpieza).getAcumula();
except
  on E : Exception do
  begin
    ERRORMATEMATICO := true;
    Evaluar := 0;
    exit;
  end;
end;

end;

// Guarda en un arreglo unidimensional el valor de las variables
// @param variable ¿Qué variable de la expresión recibirá el valor? a..z
// @param valor  Valor de tipo double que tendrá esa variable.
procedure TEvaluadorExpresionAlgebraica.DarValorVariable(variable: char; valor: double);
begin
        valorVariable[ord(variable) - ASCIILETRA] := valor;
end;

// Retorna true si hubo un error matemático
// @return TRUE si hubo un error matemático
function TEvaluadorExpresionAlgebraica.getErrorMatematico(): boolean;
begin
  getErrorMatematico := ERRORMATEMATICO;
end;

//Libera la memoria
procedure TEvaluadorExpresionAlgebraica.LiberaMemoria();
begin
  Piezas.Free;
end;

end.
```

PARTE 6. PROBANDO EL EVALUADOR DE EXPRESIONES

C++

Puede hacer uso de la herramienta libre Dev-C++ que se puede obtener en http://www.bloodshed.net/devcpp.html o de Microsoft Visual Studio C++ Express 2010 en: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express
A continuación, se presenta una serie de capturas de pantallas de cómo usar estos entornos de programación paso a paso para poder hacer uso del evaluador de expresiones.

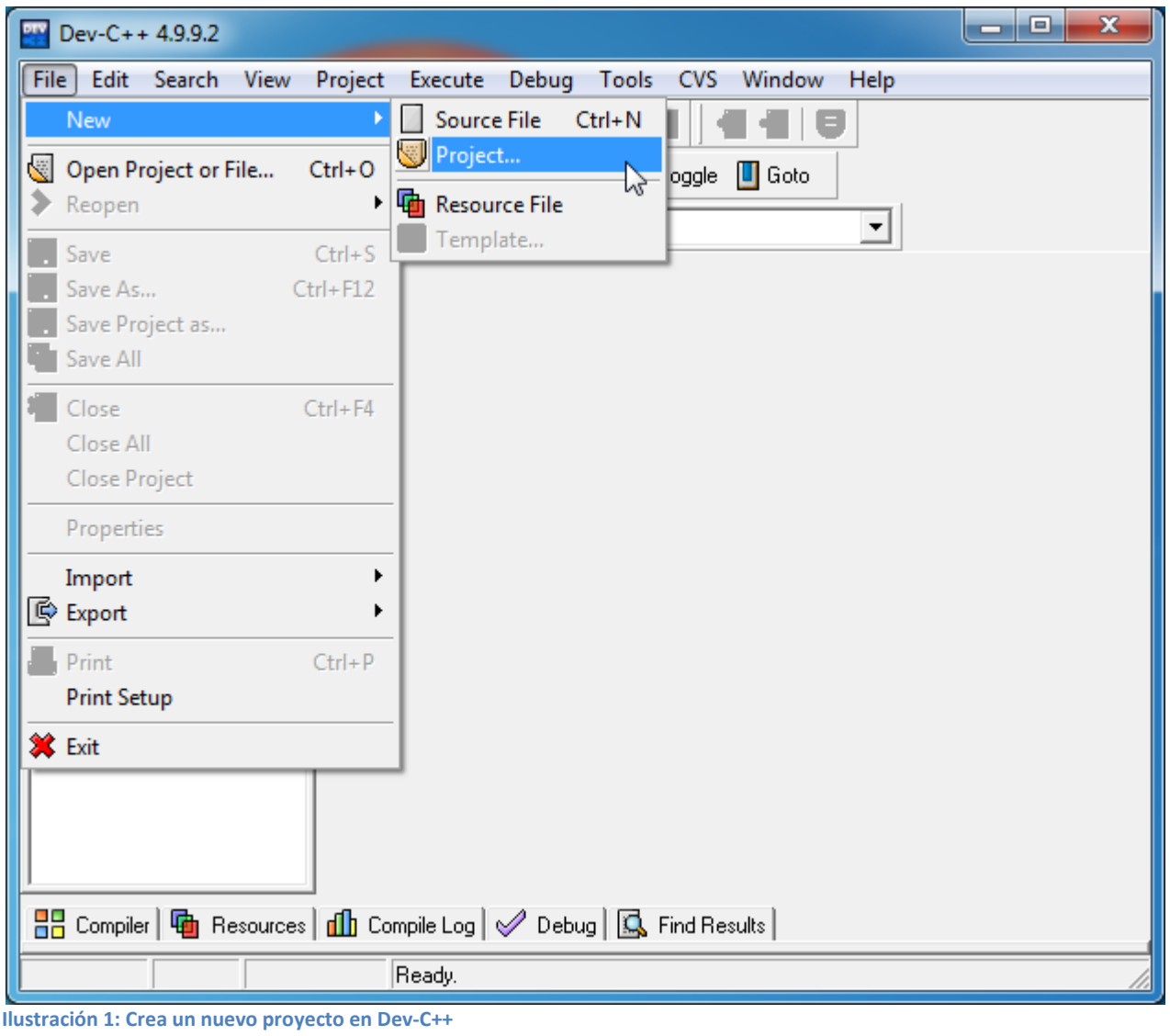


Ilustración 1: Crea un nuevo proyecto en Dev-C++

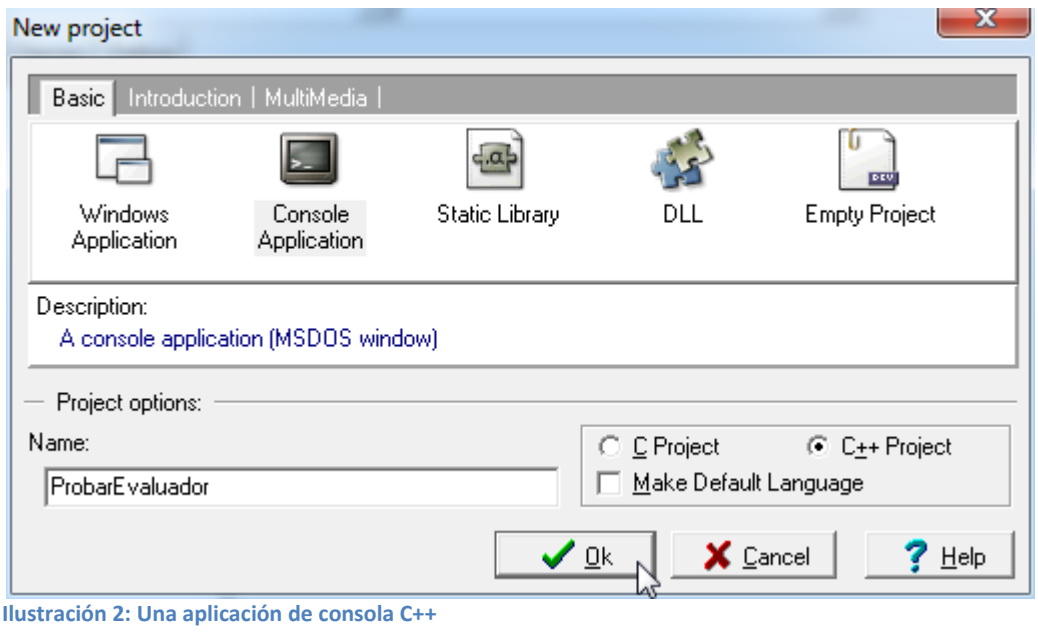


Ilustración 2: Una aplicación de consola C++

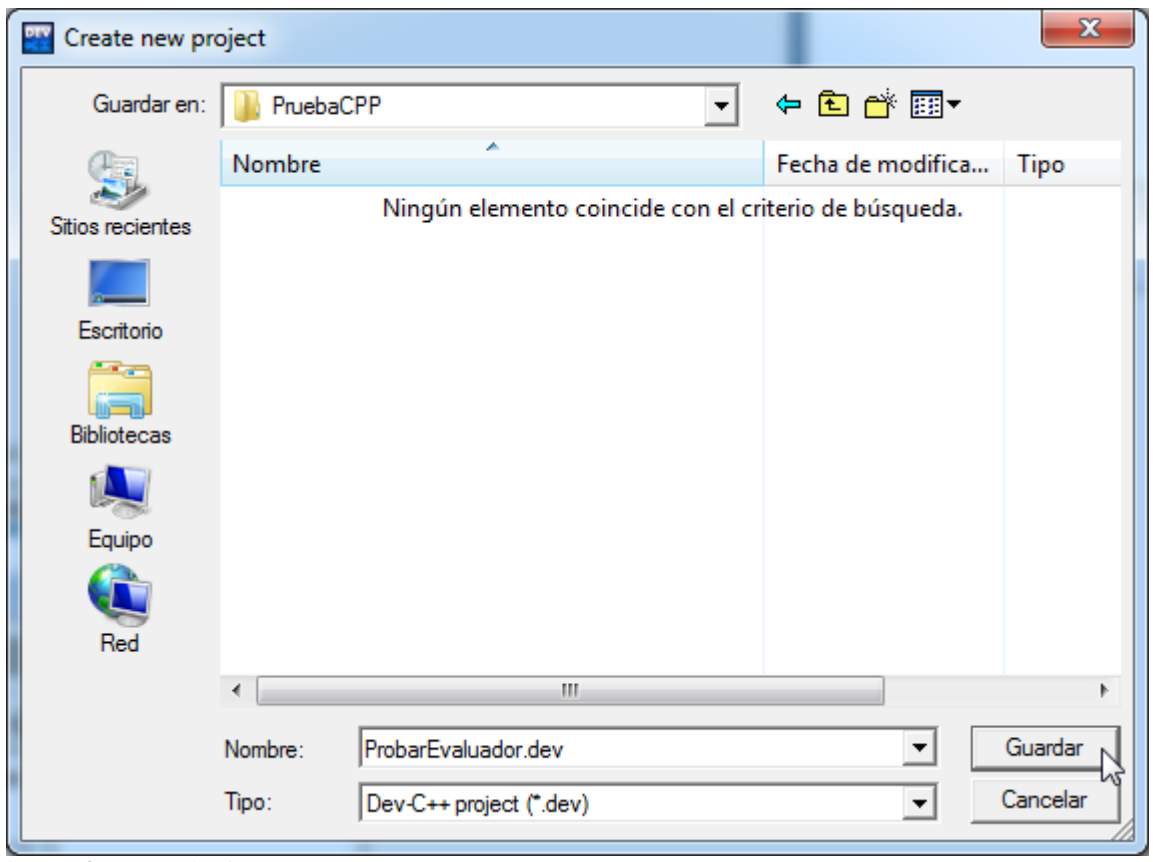


Ilustración 3: Guarda el proyecto

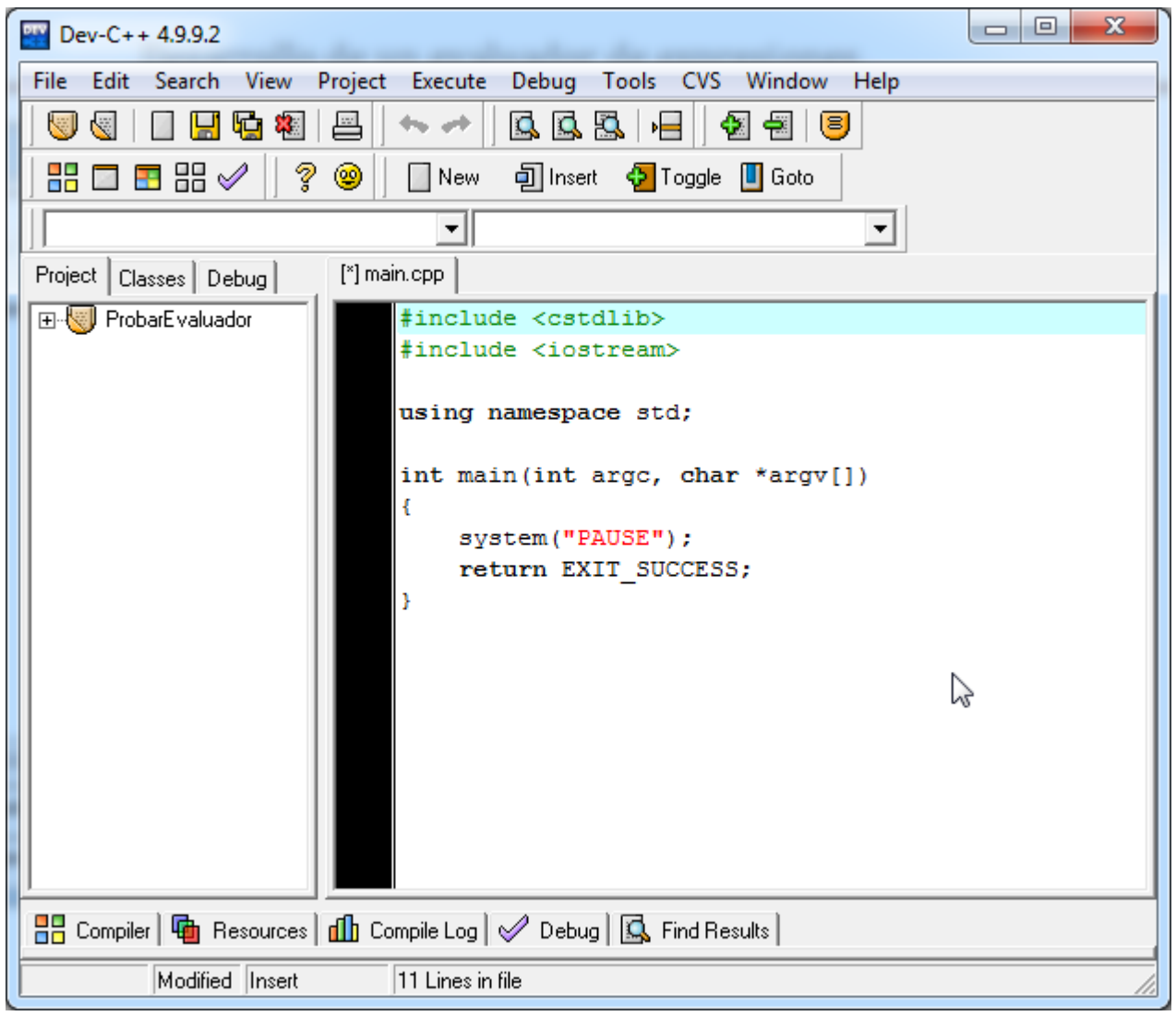


Ilustración 4: Se genera un archivo de inicio por defecto

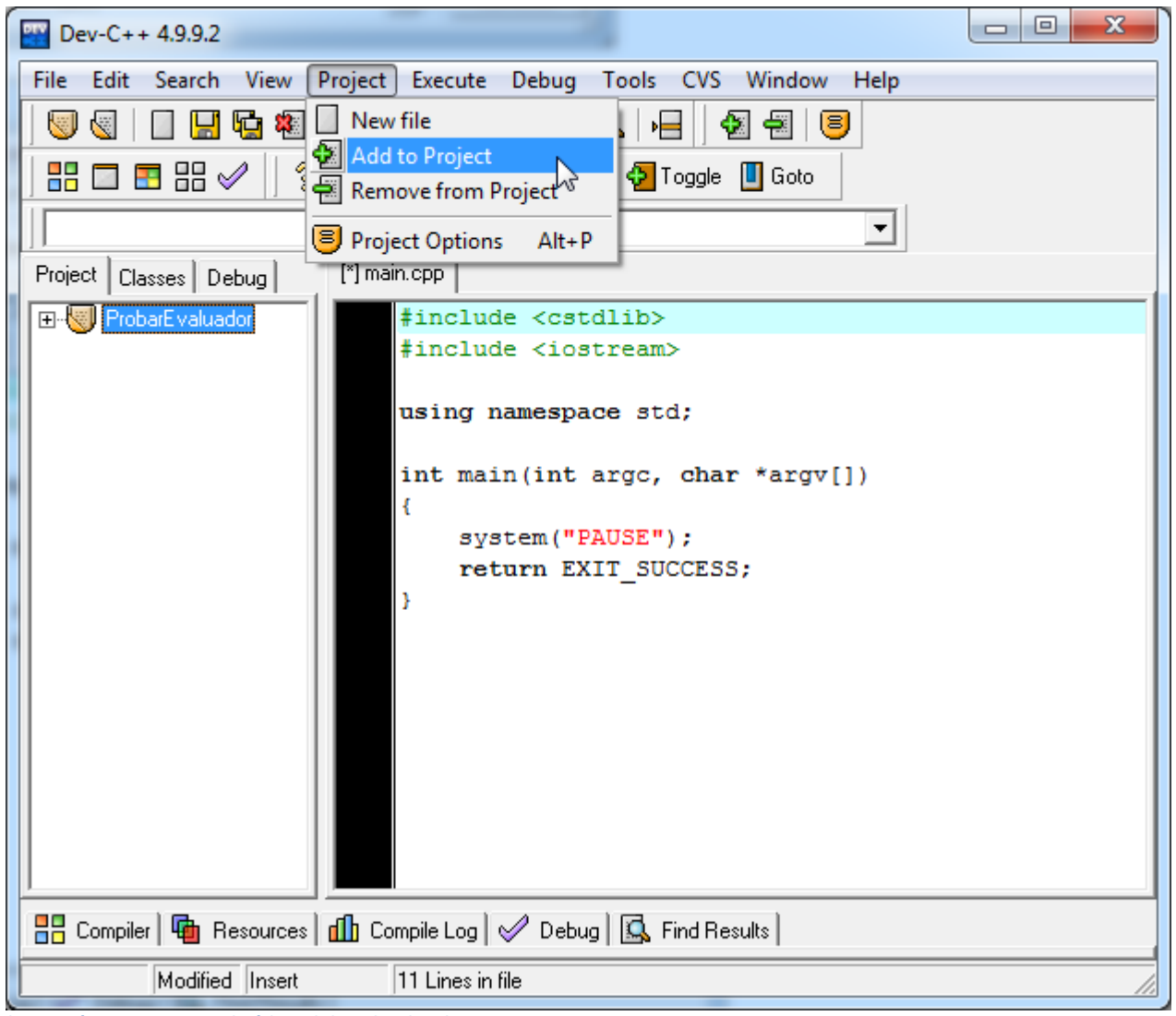


Ilustración 5: Se agrega el código del evaluador de expresiones

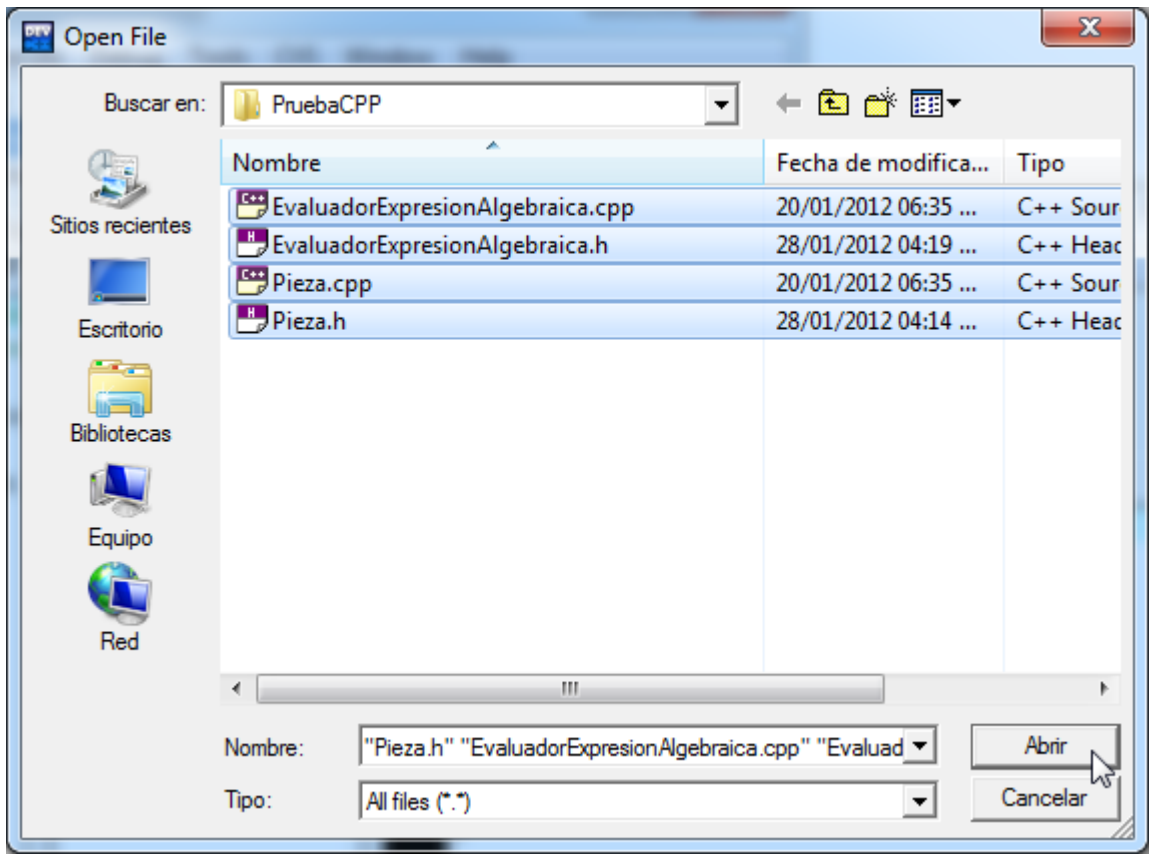


Ilustración 6: En C++ son 4 archivos

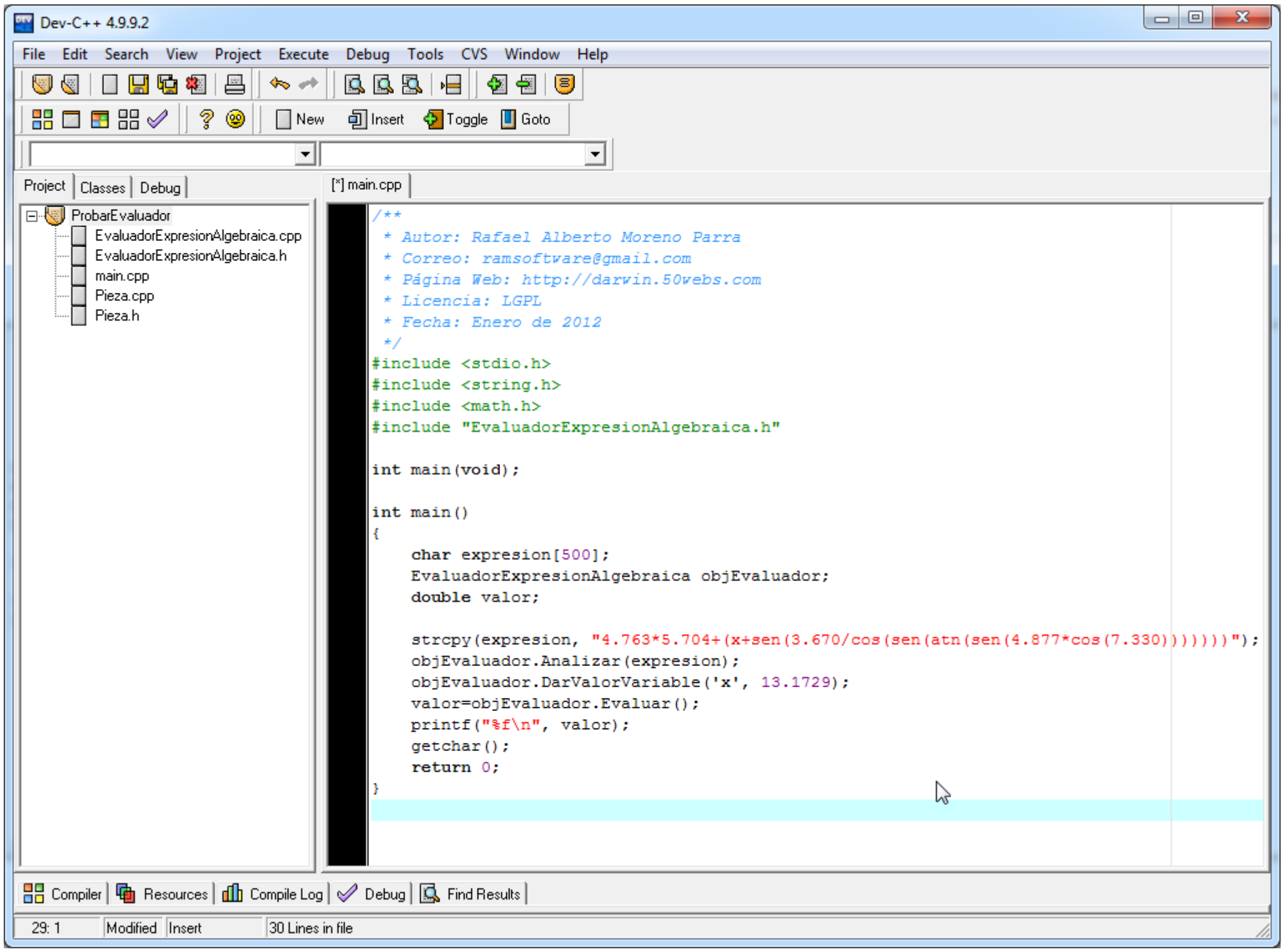


Ilustración 7: Se escribe el código del archivo de inicialización

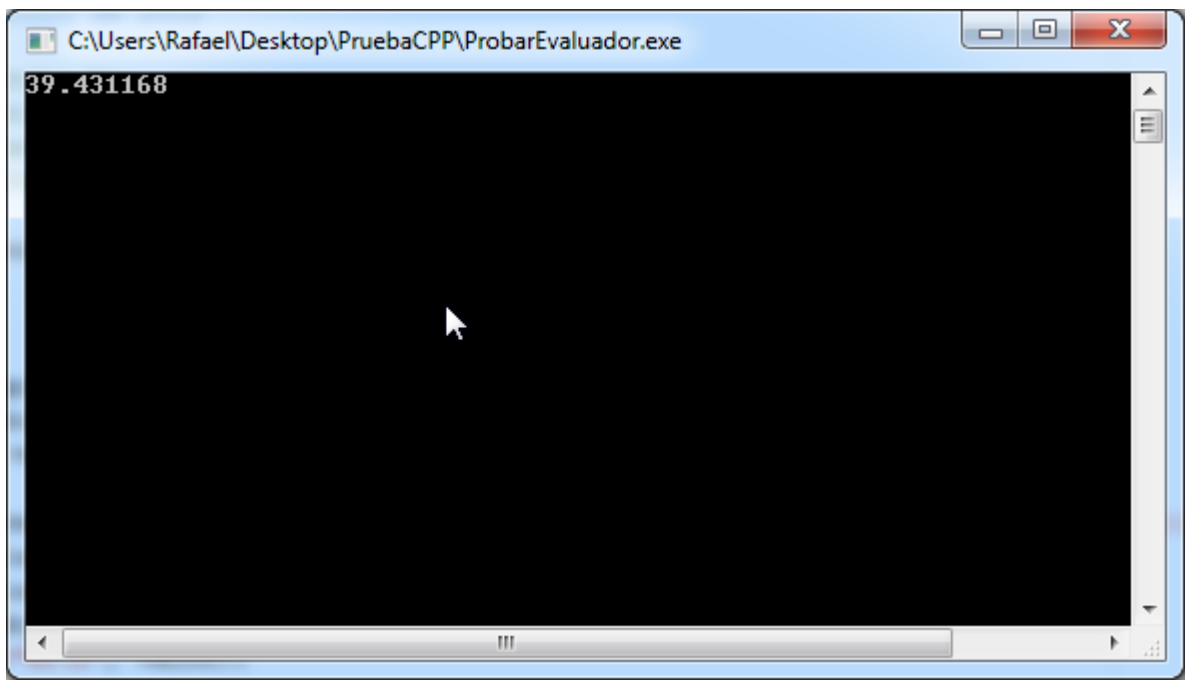


Ilustración 8: Ejecución del código de ejemplo

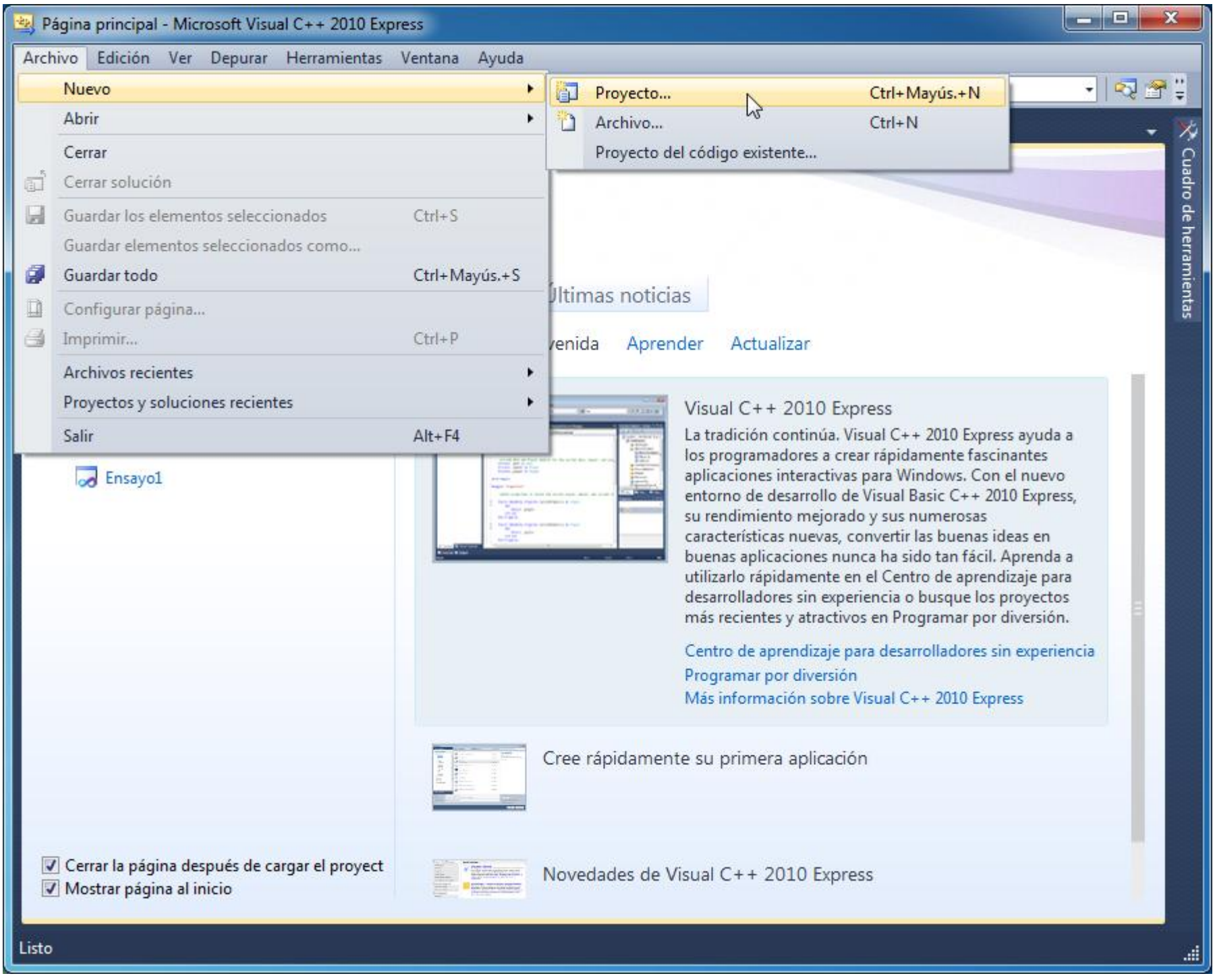


Ilustración 9: En Microsoft Visual C++ 2010

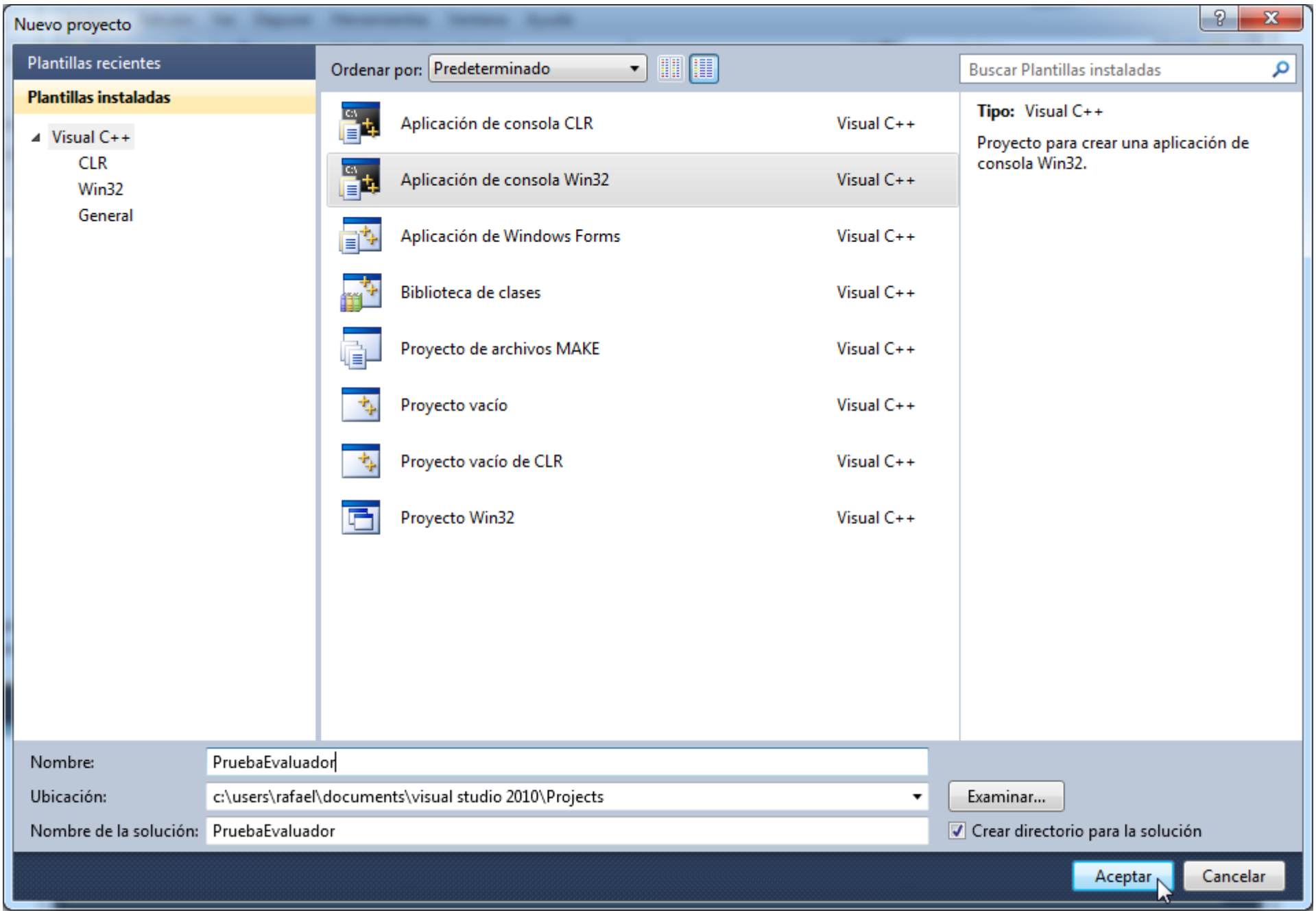


Ilustración 10: Se genera un nuevo proyecto de consola Win32

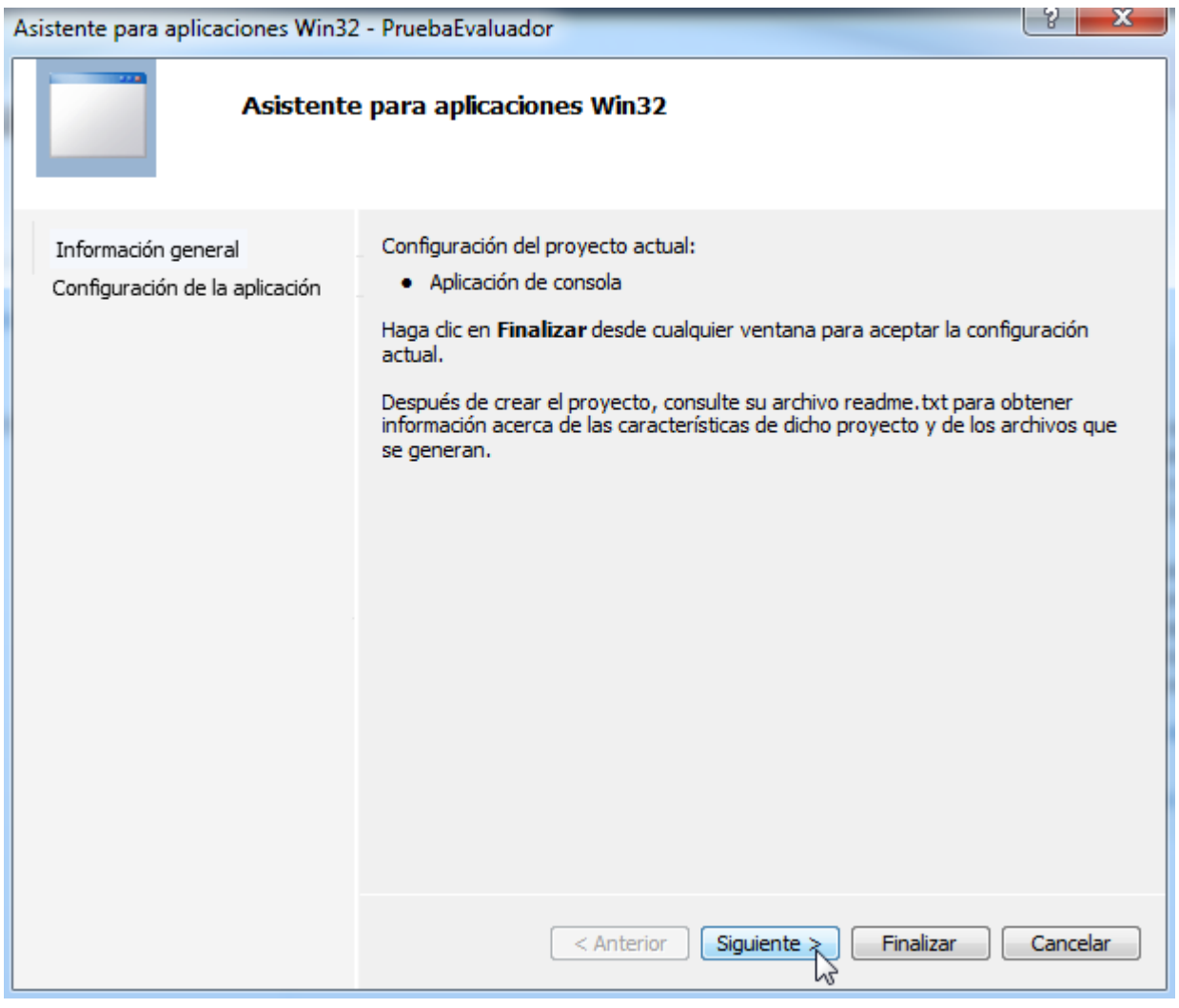


Ilustración 11: Este es el asistente para generar aplicaciones de consola

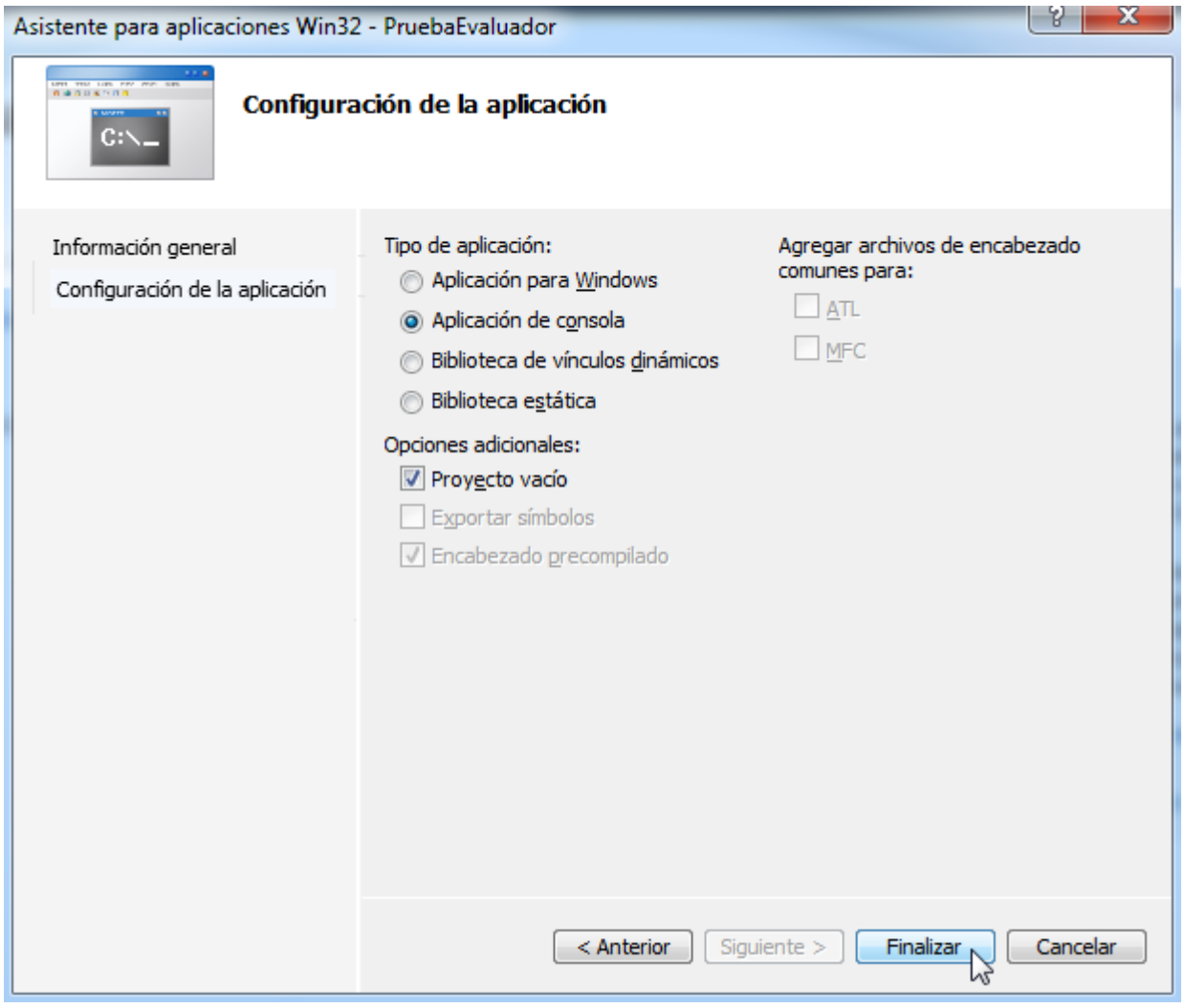


Ilustración 12: Se genera un proyecto vacío

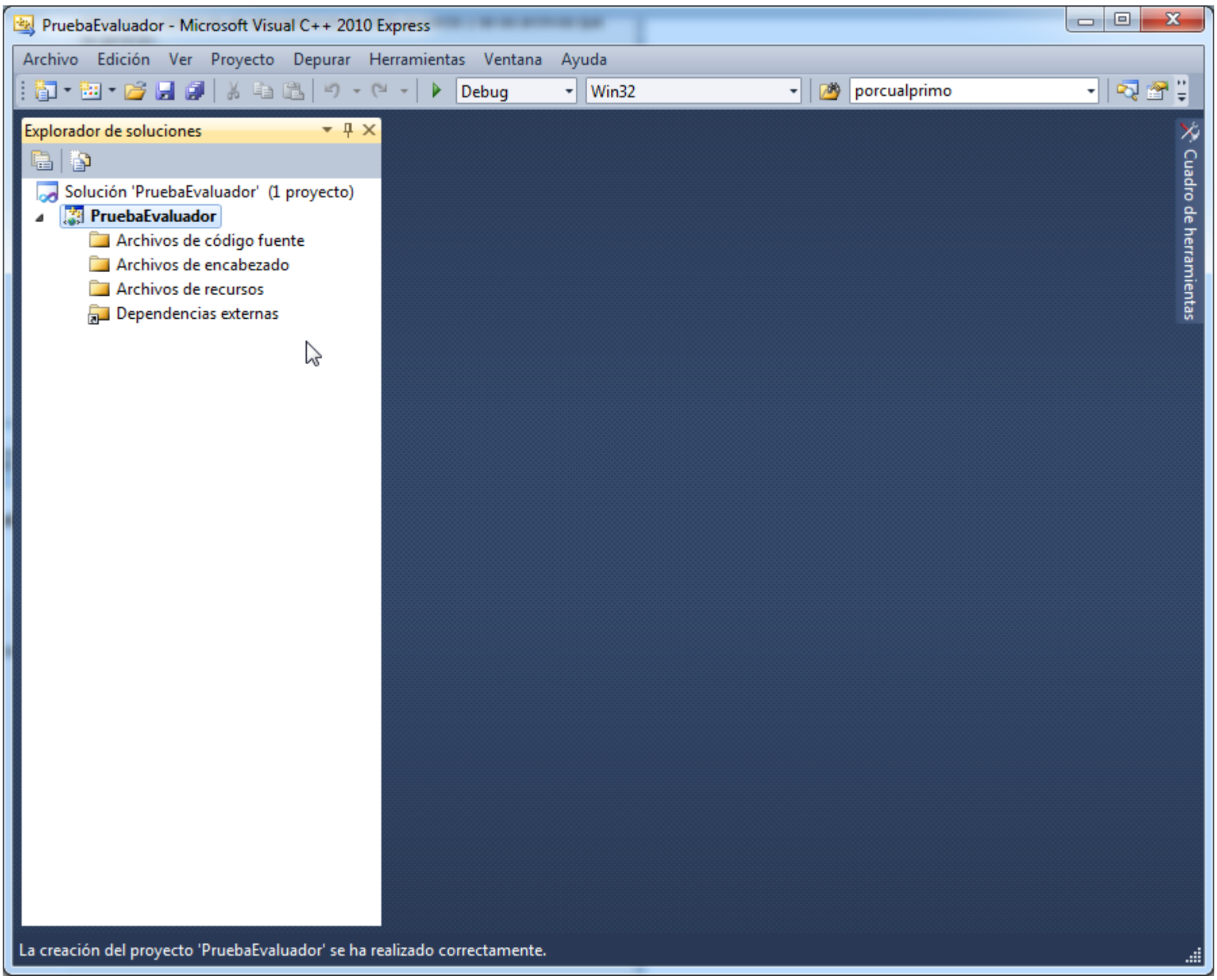


Ilustración 13: Así luce un proyecto vacío

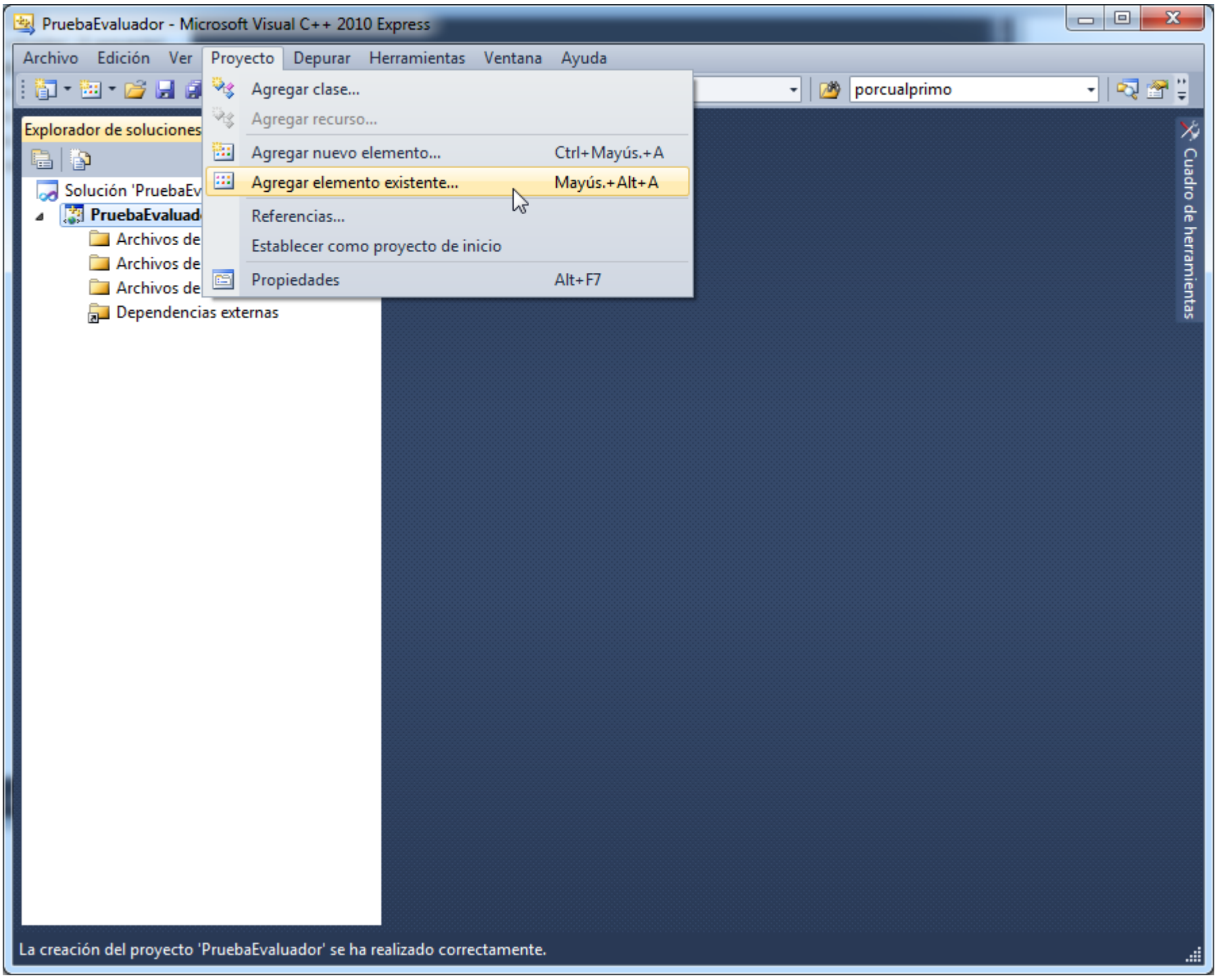


Ilustración 14: Agrega los archivos del evaluador de expresiones

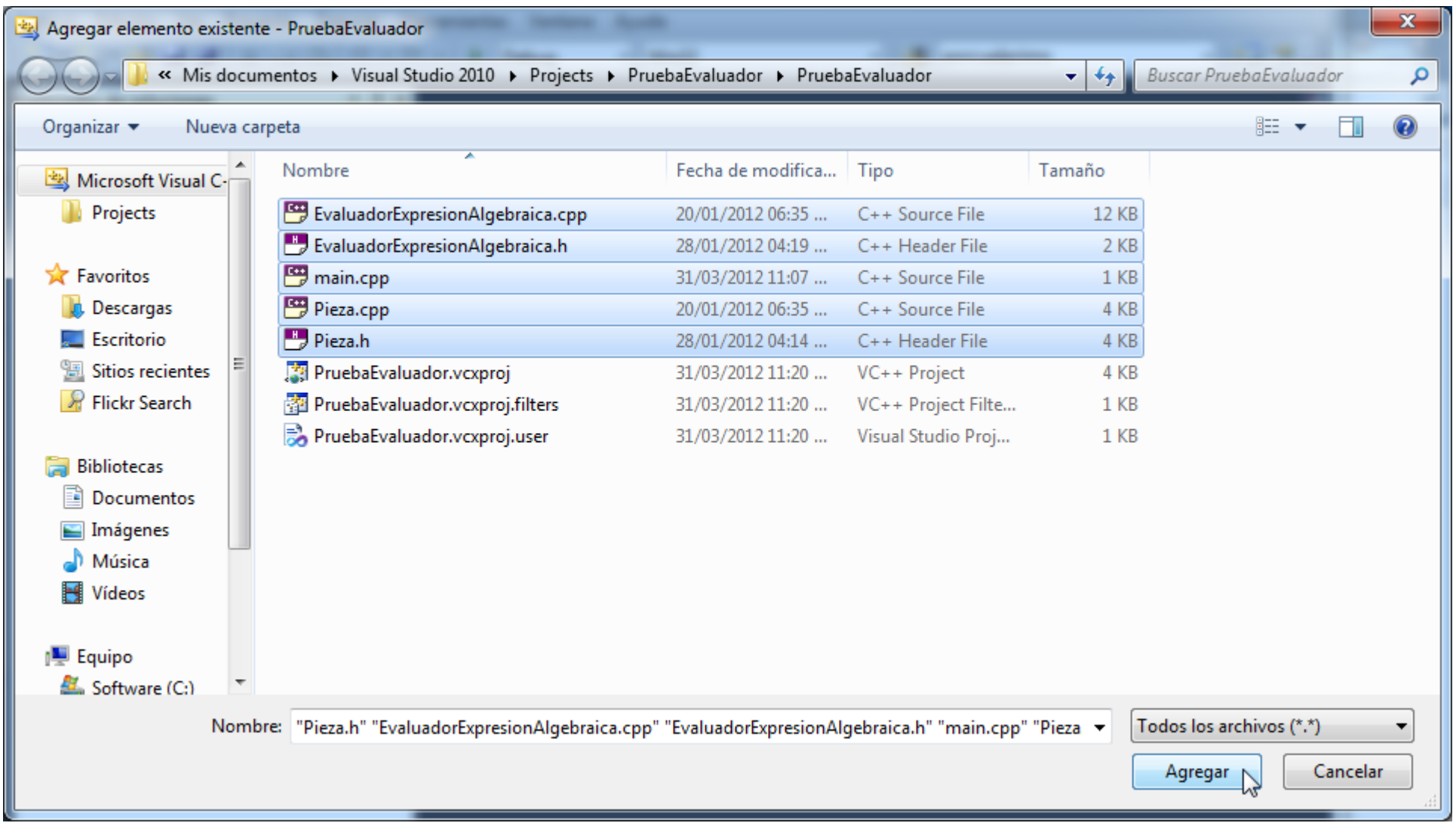


Ilustración 15: Son cinco(5) archivos, incluye el archivo de inicio o principal

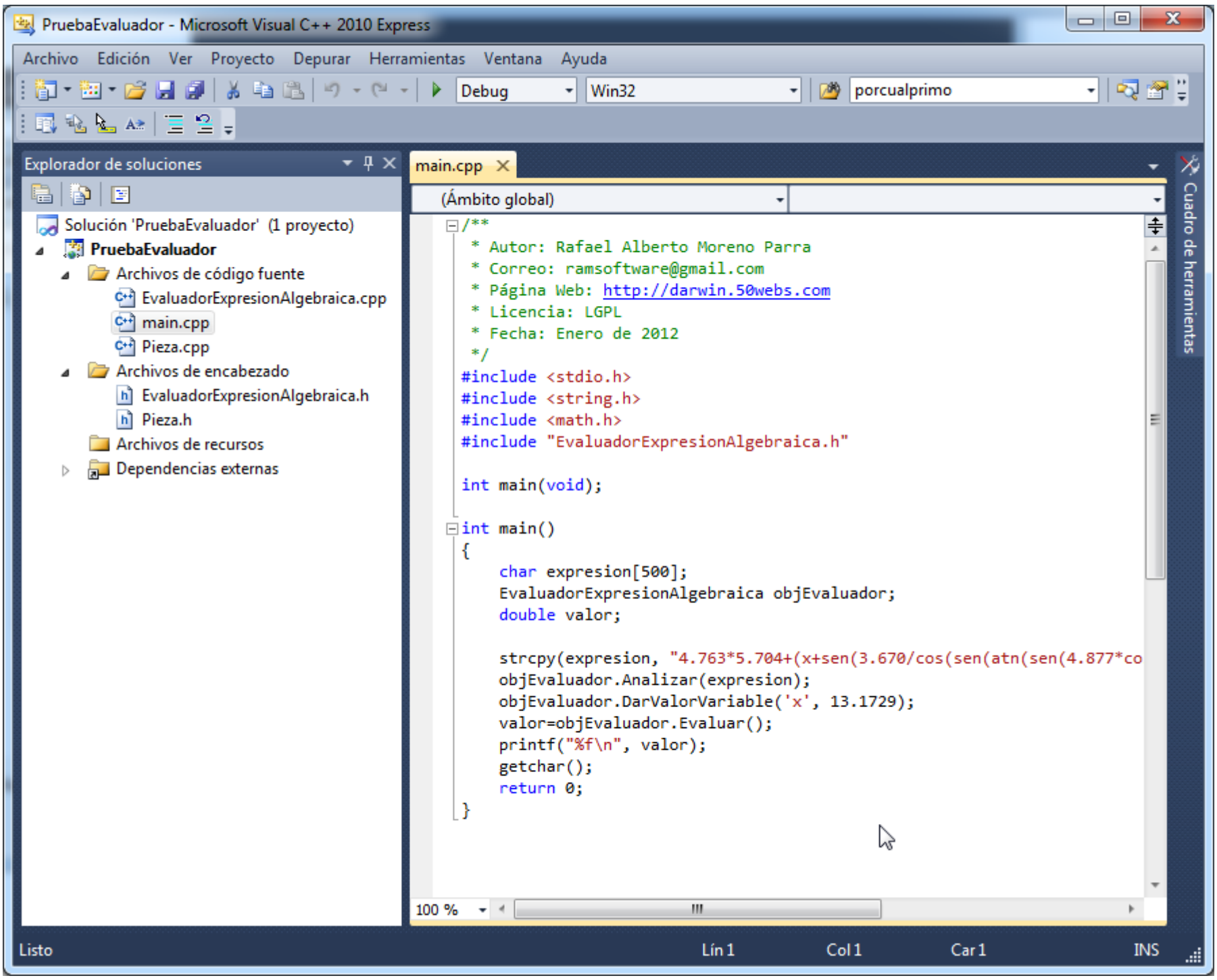


Ilustración 16: Se observa el archivo de inicio y el código necesario para probar el evaluador de expresiones

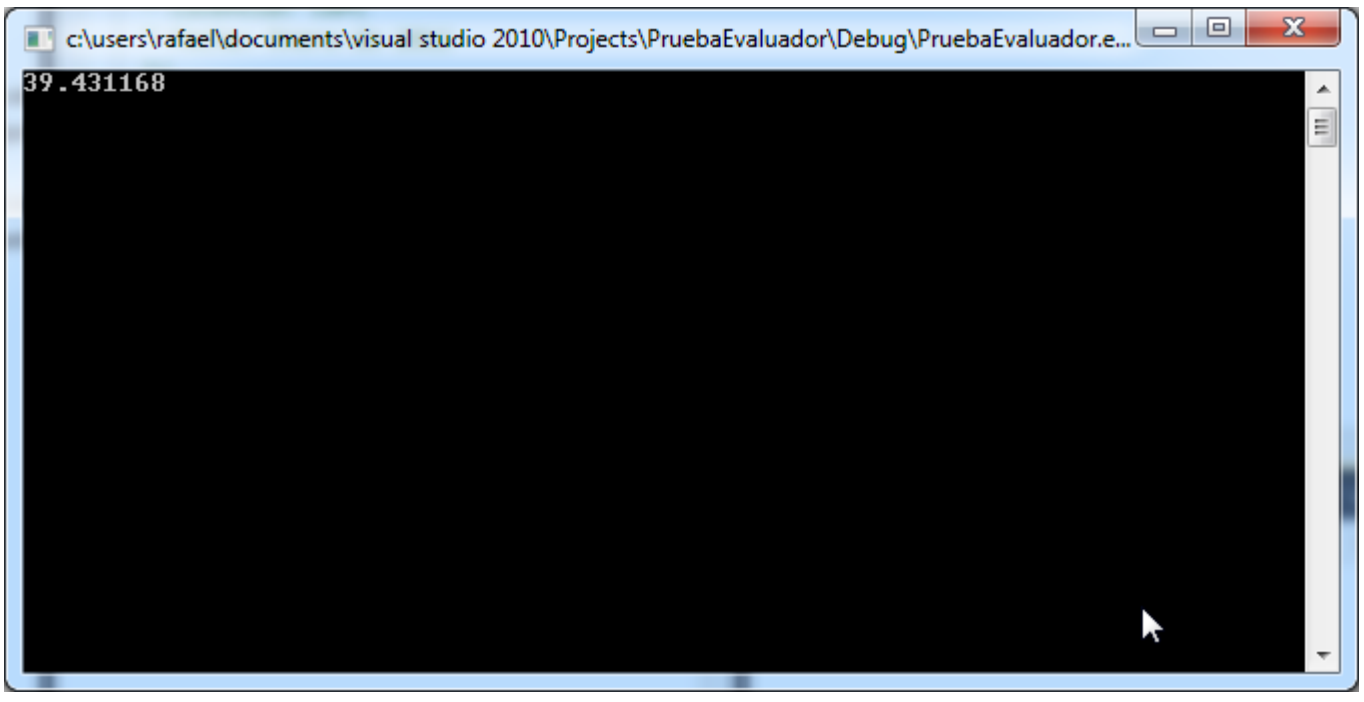


Ilustración 17: Ejecución del proyecto

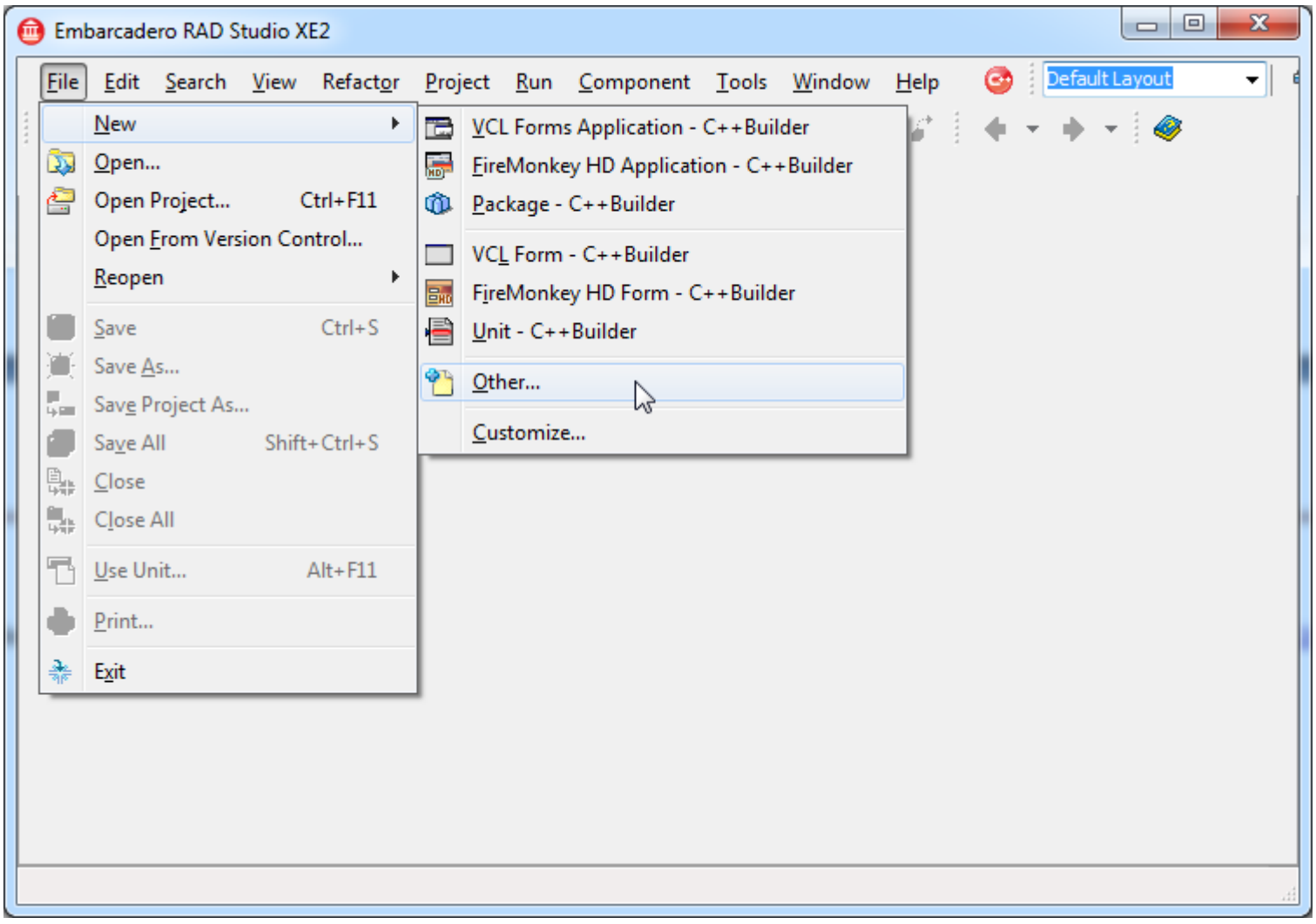


Ilustración 18: Probando en Embarcadero RAD Studio XE2

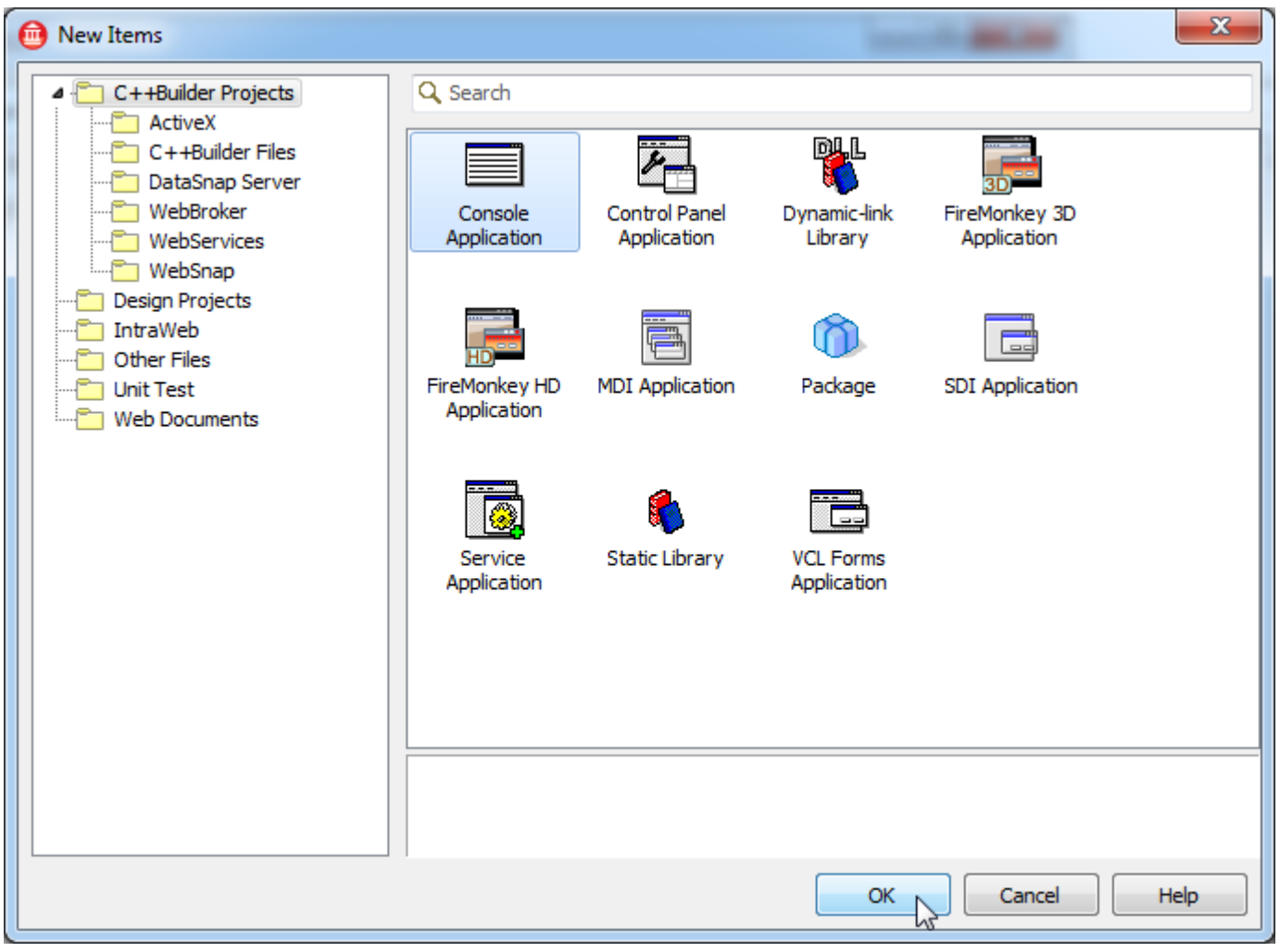


Ilustración 19: Busca una aplicación de consola

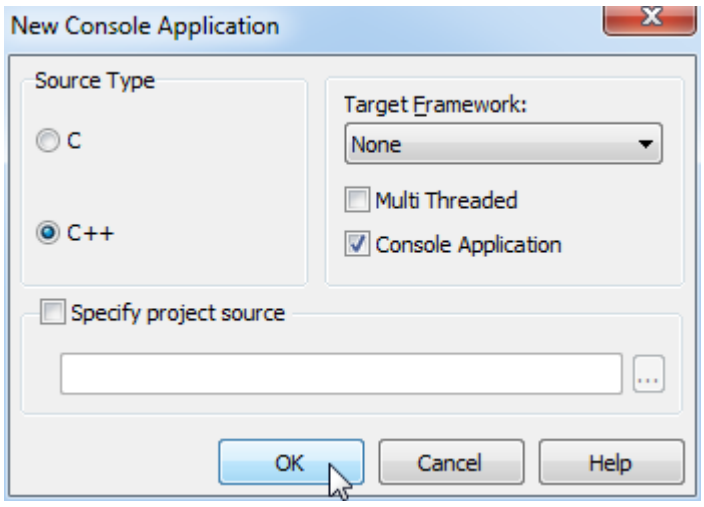


Ilustración 20: Una aplicación de consola en C++

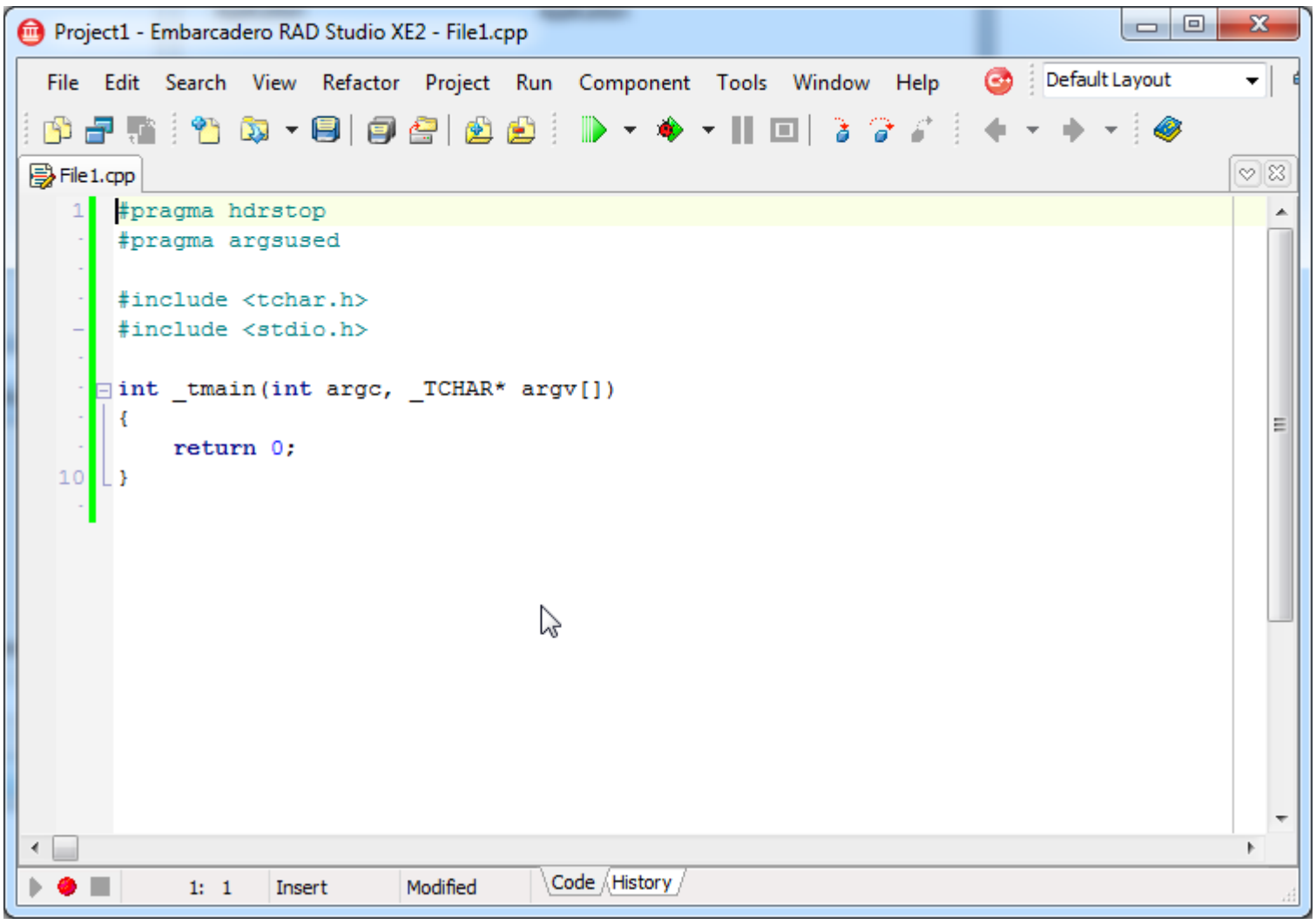


Ilustración 21: Código por defecto

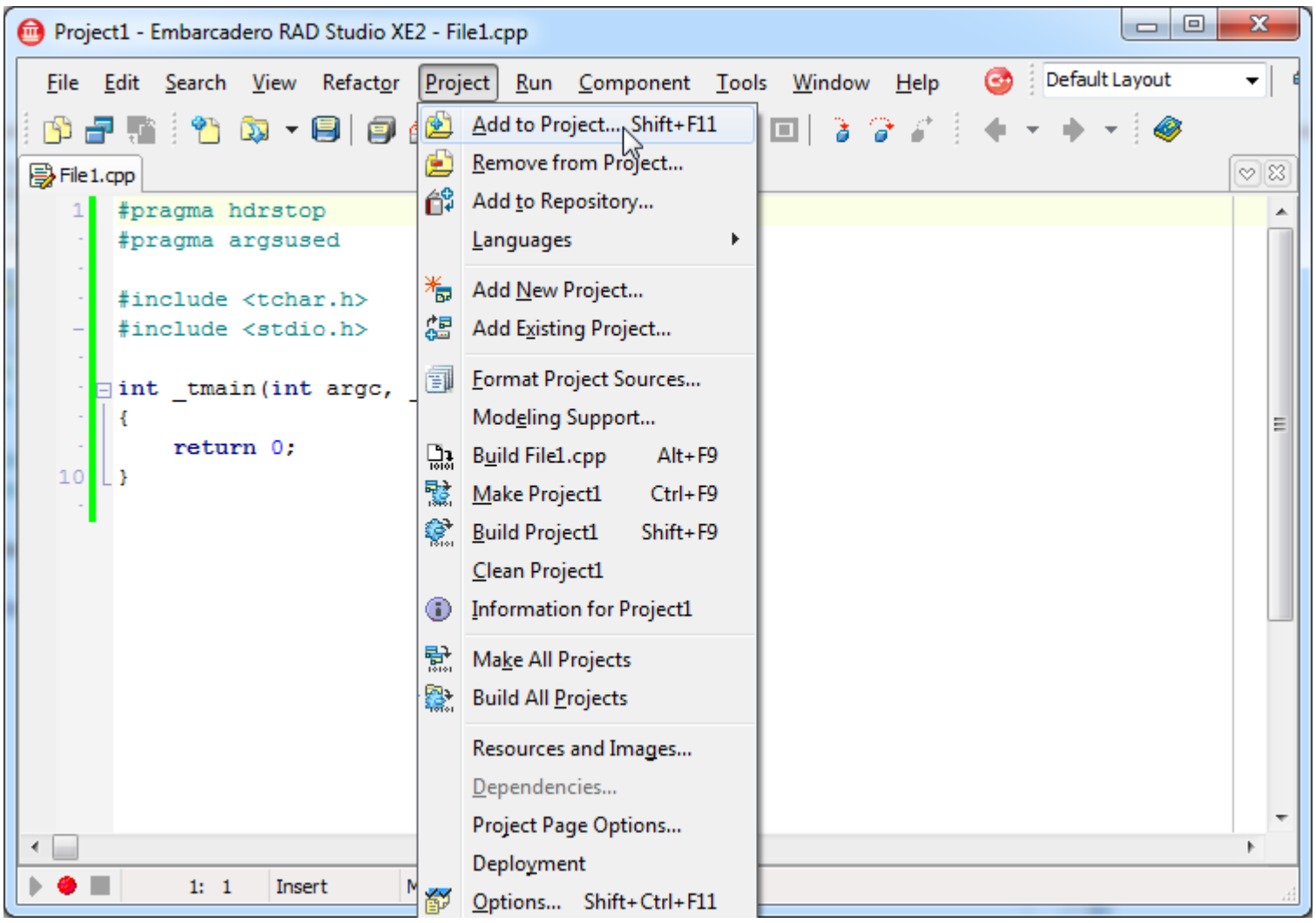


Ilustración 22: Se agrega el código del evaluador de expresiones

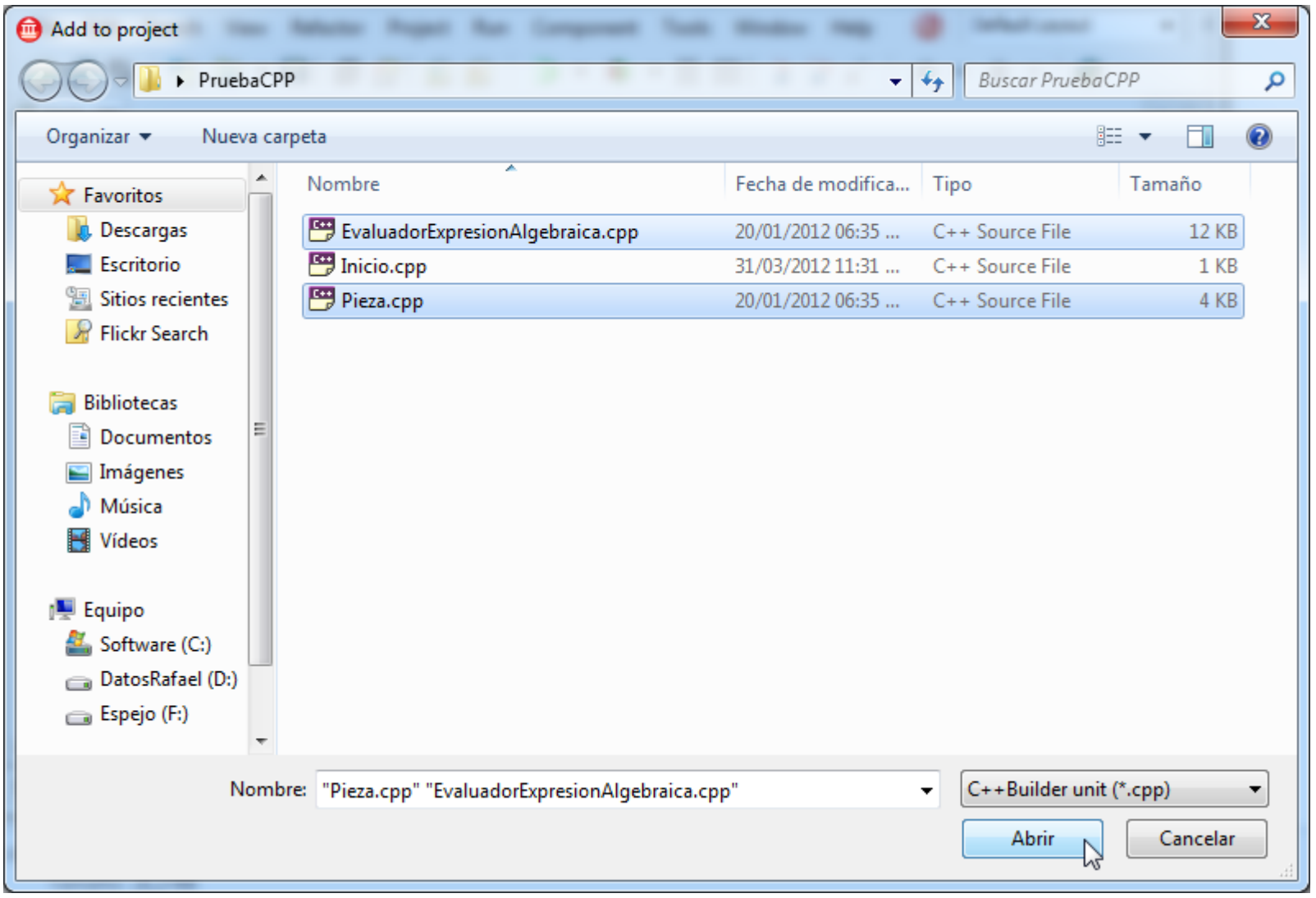


Ilustración 23: El código en C++ (*.cpp)

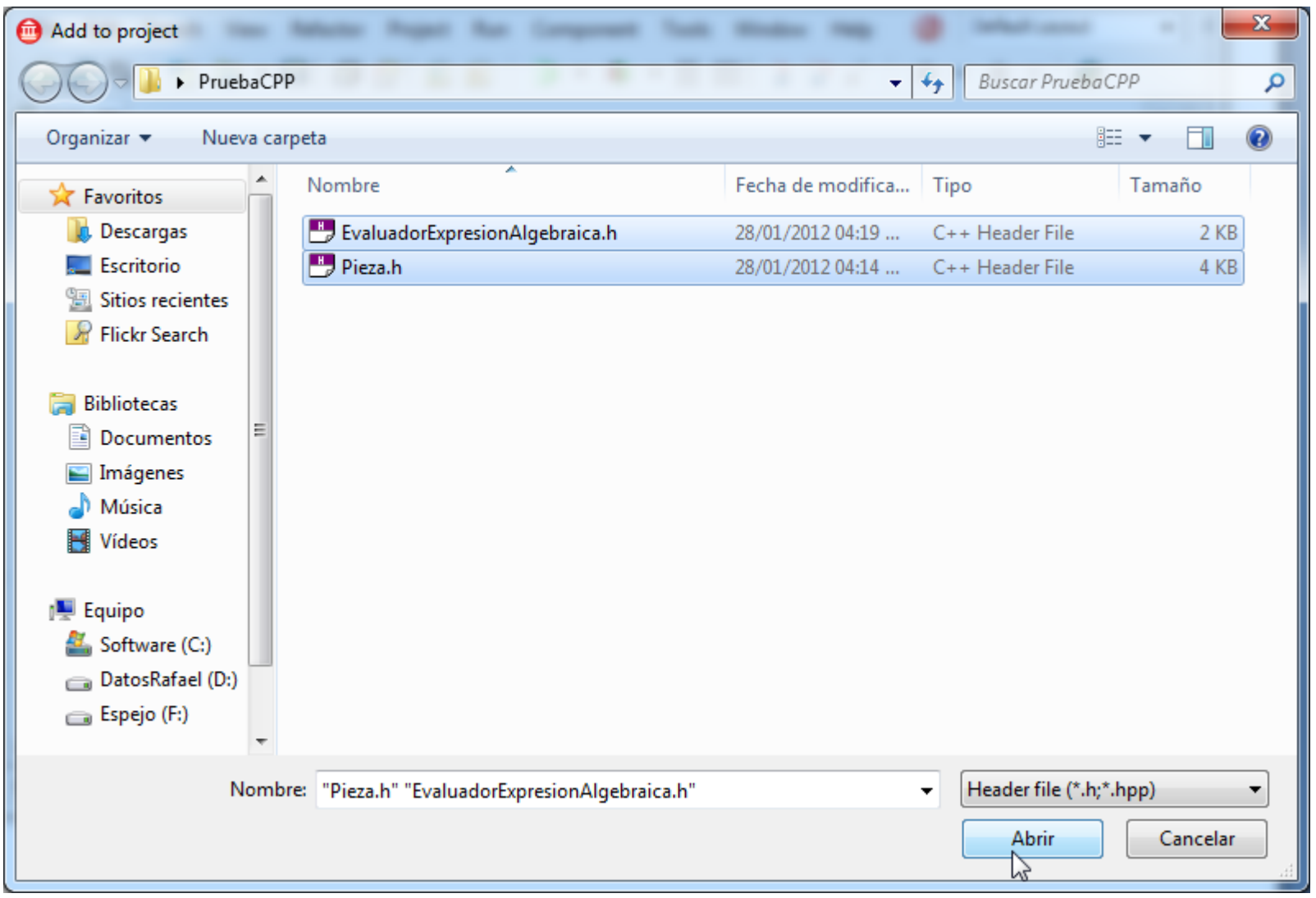


Ilustración 24: Se agrega también los archivos de encabezado (*.h)

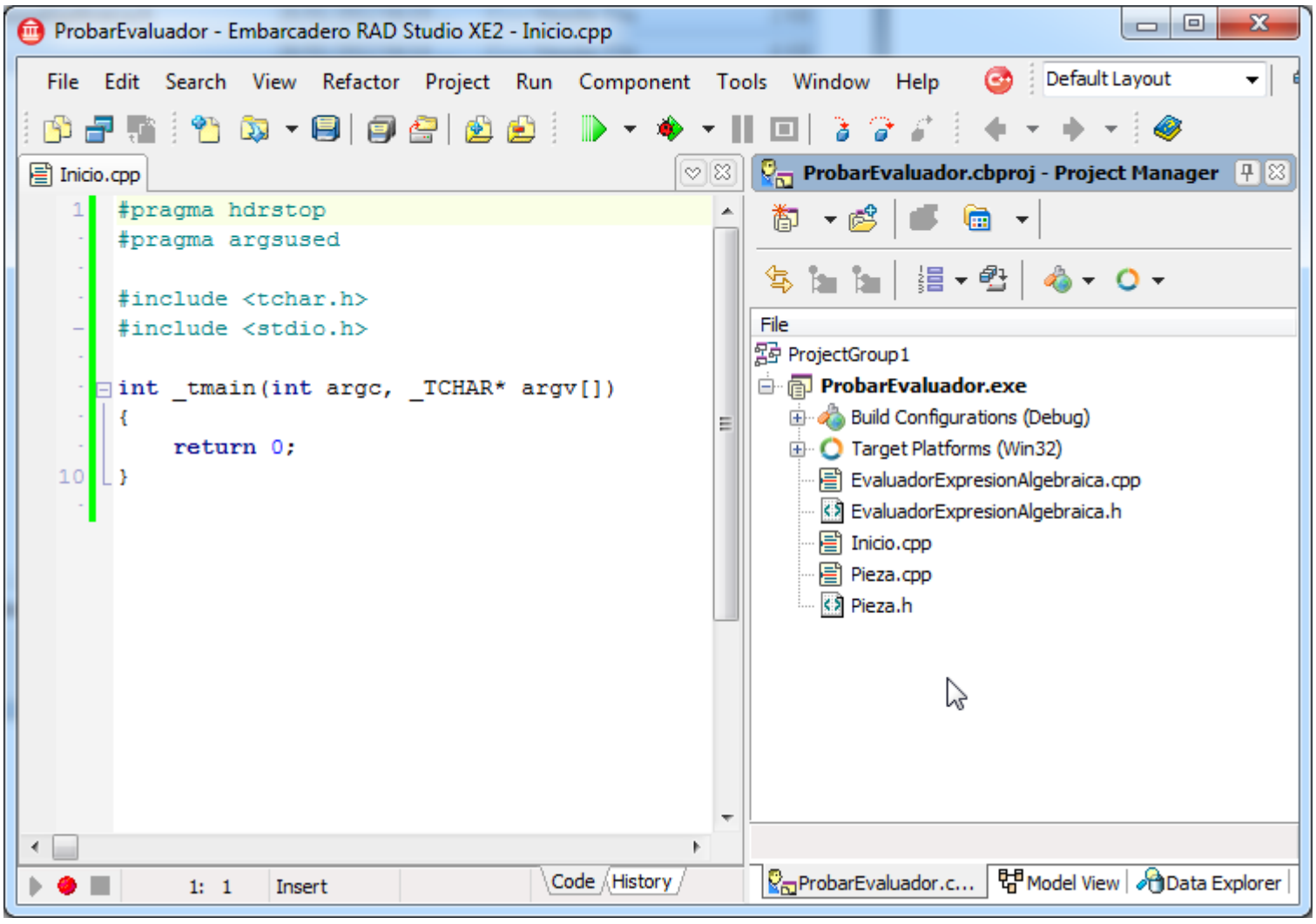


Ilustración 25: En la ventana de "Project Manager" se observa como se arma el proyecto

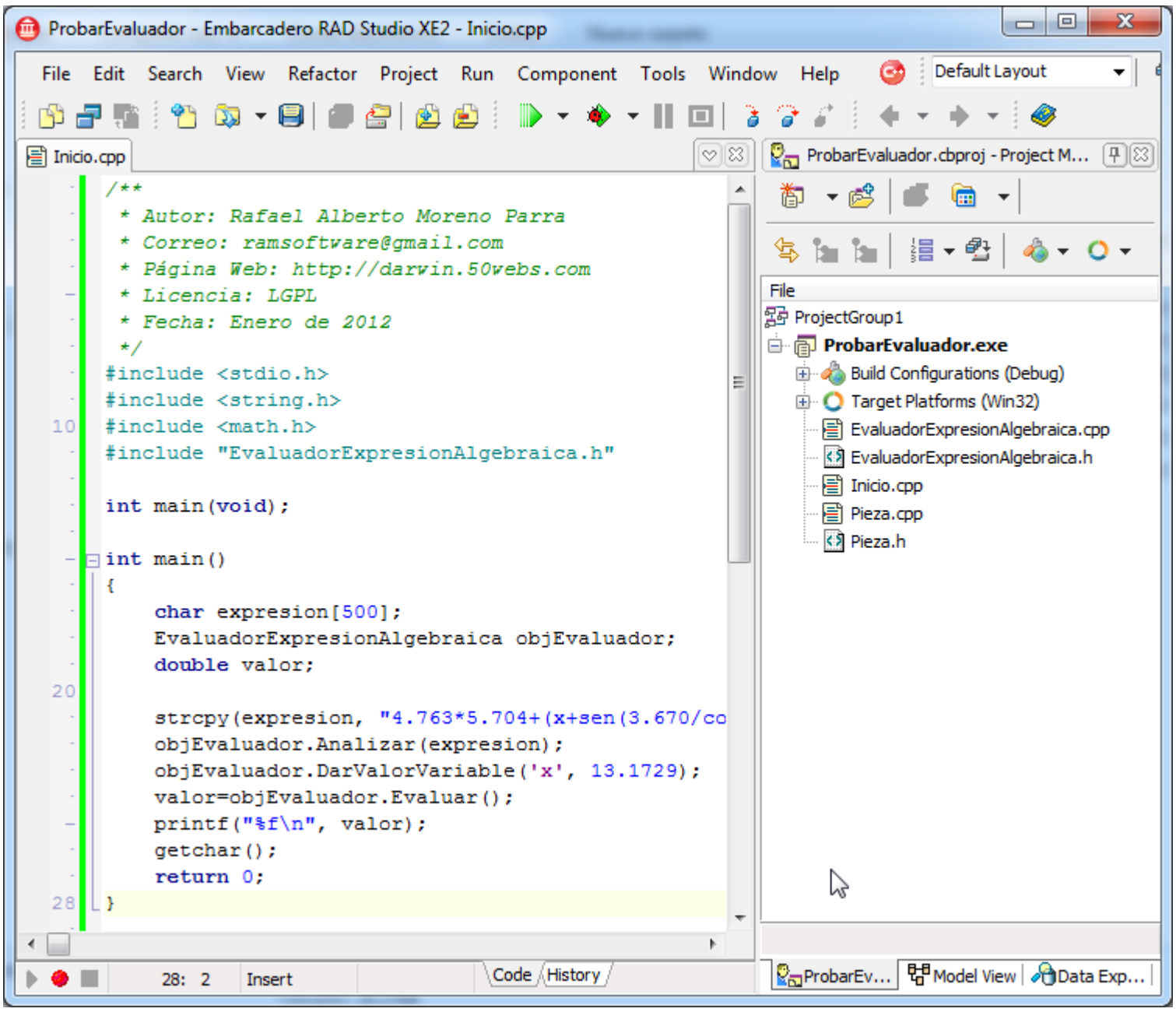


Ilustración 26: Se observa el código de inicio para probar el evaluador

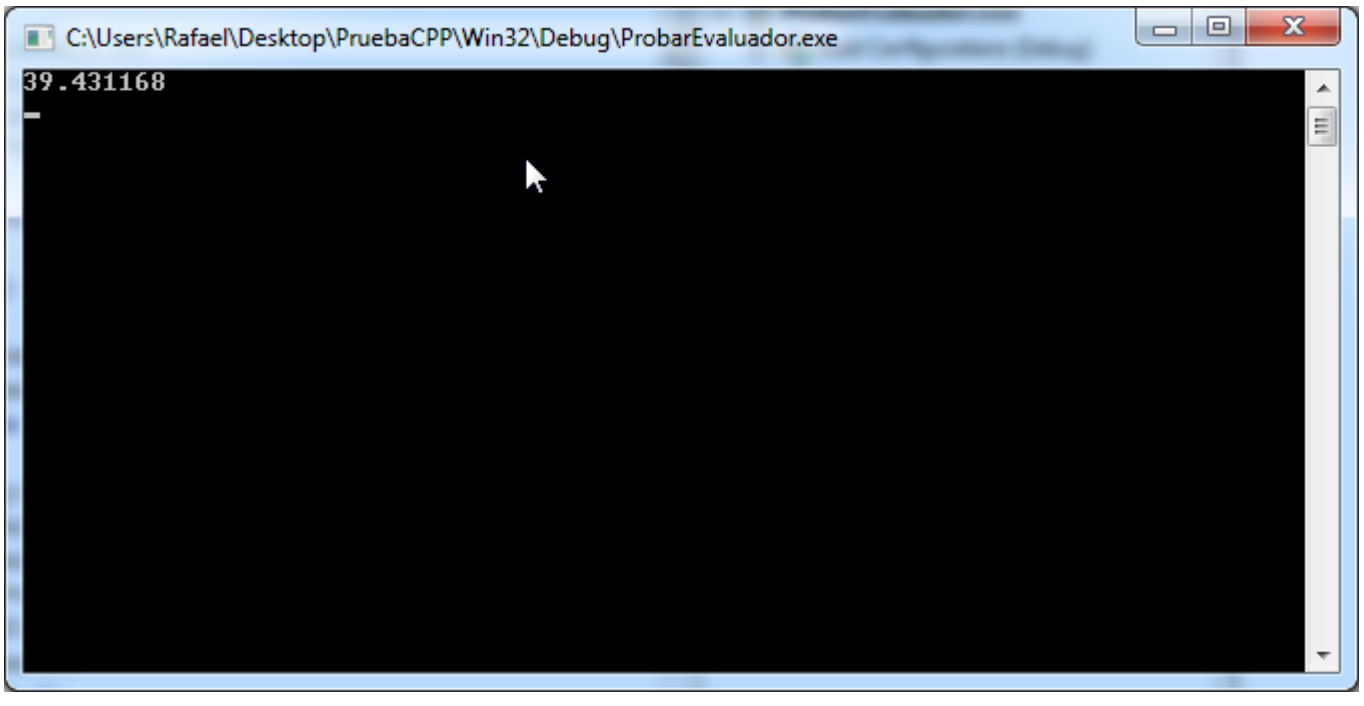


Ilustración 27: Ejecución del código

Visual C#

Puede hacer uso de Microsoft Visual C# Express 2010 (es gratuita). En: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express

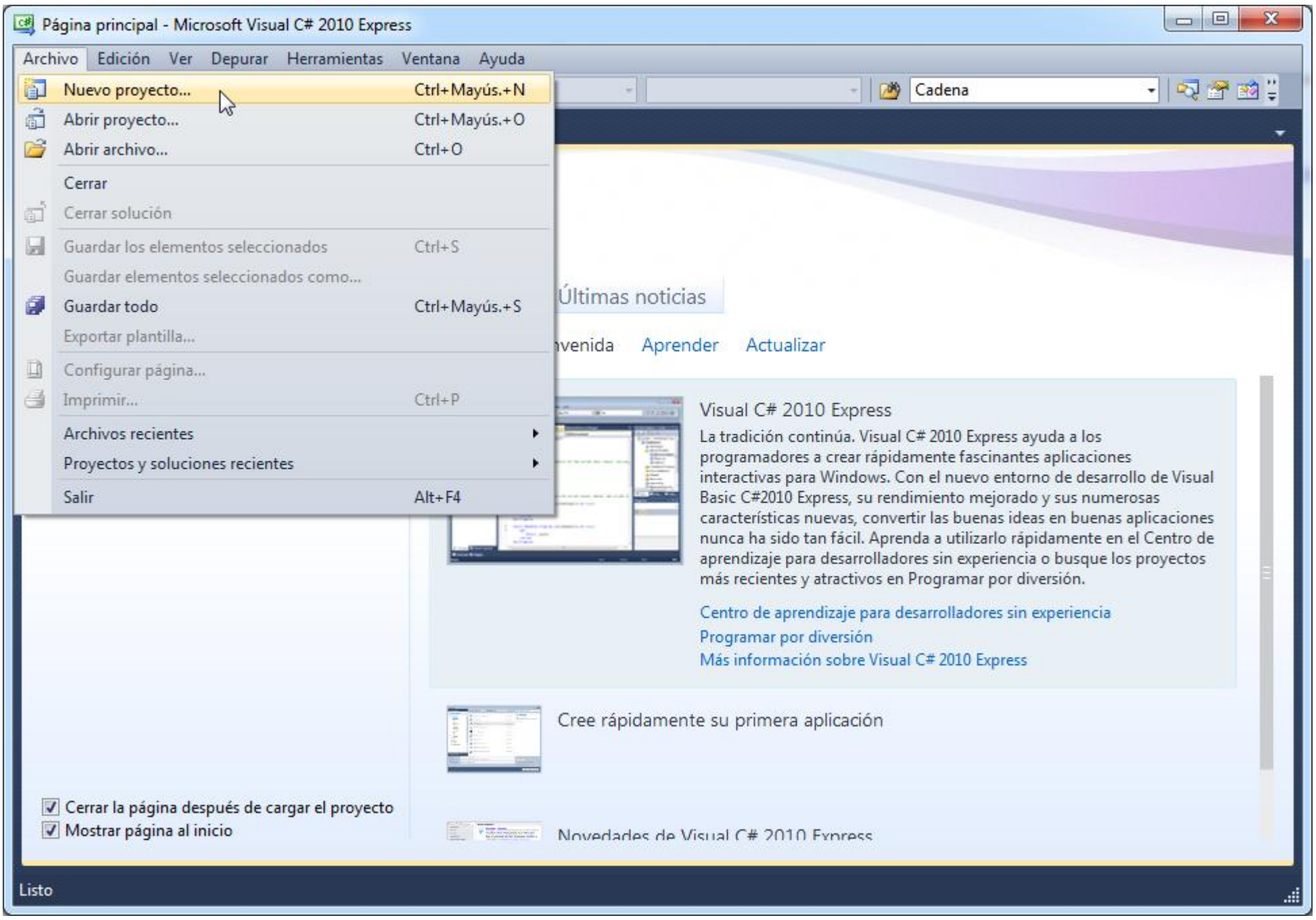


Ilustración 28: Se genera un nuevo proyecto en Visual C# 2010

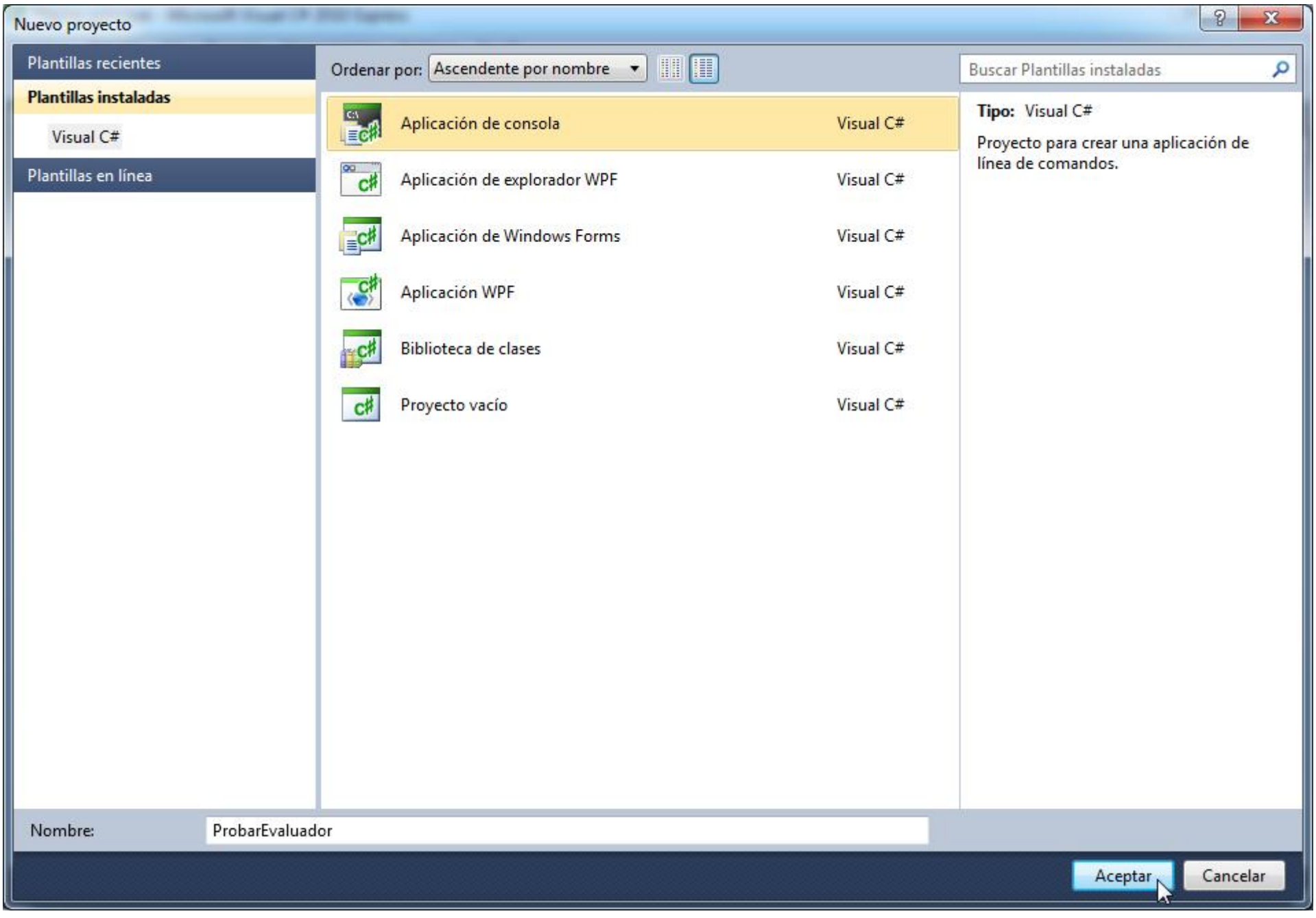


Ilustración 29: Aplicación de consola de Visual C#

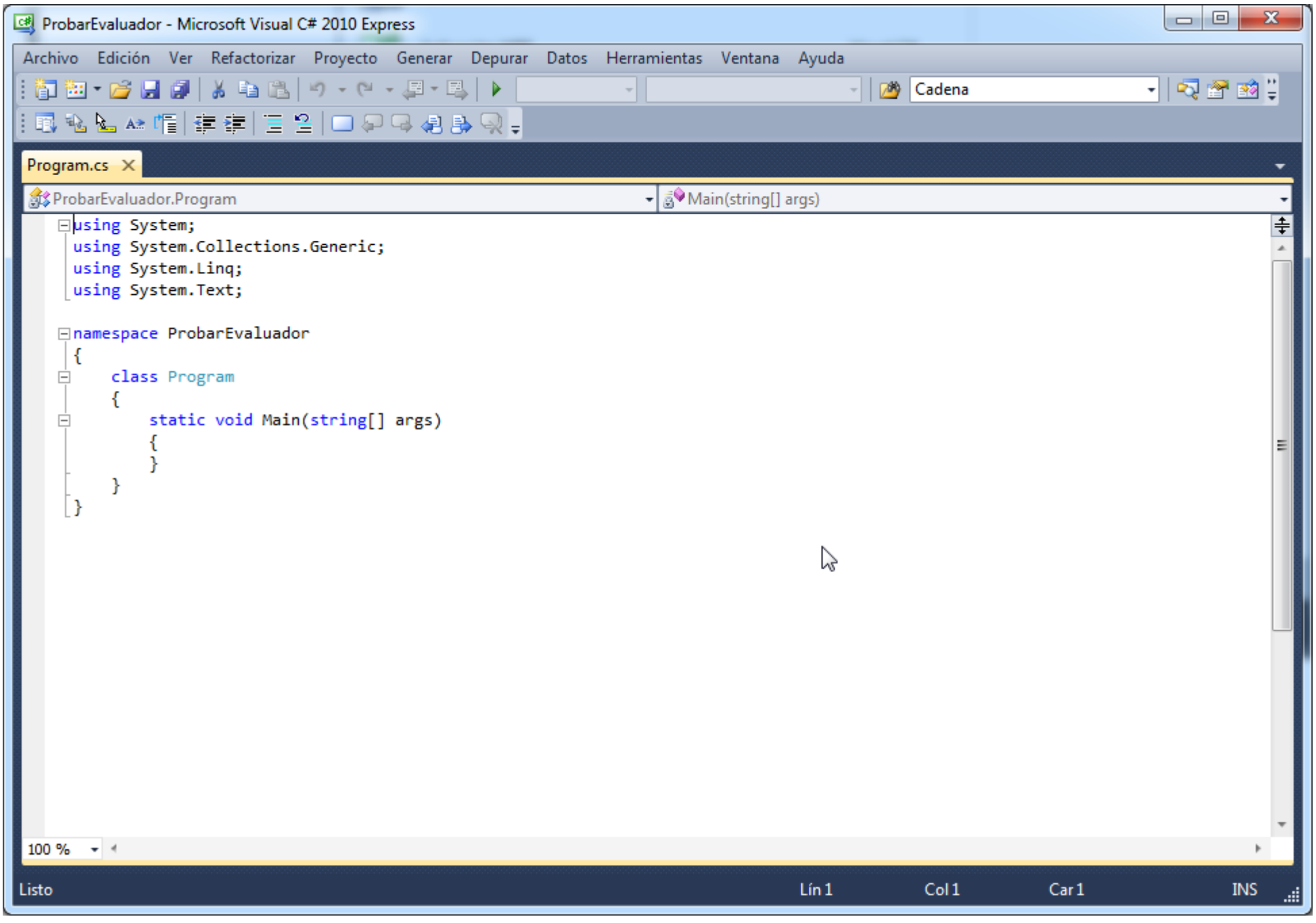


Ilustración 30: Código por defecto

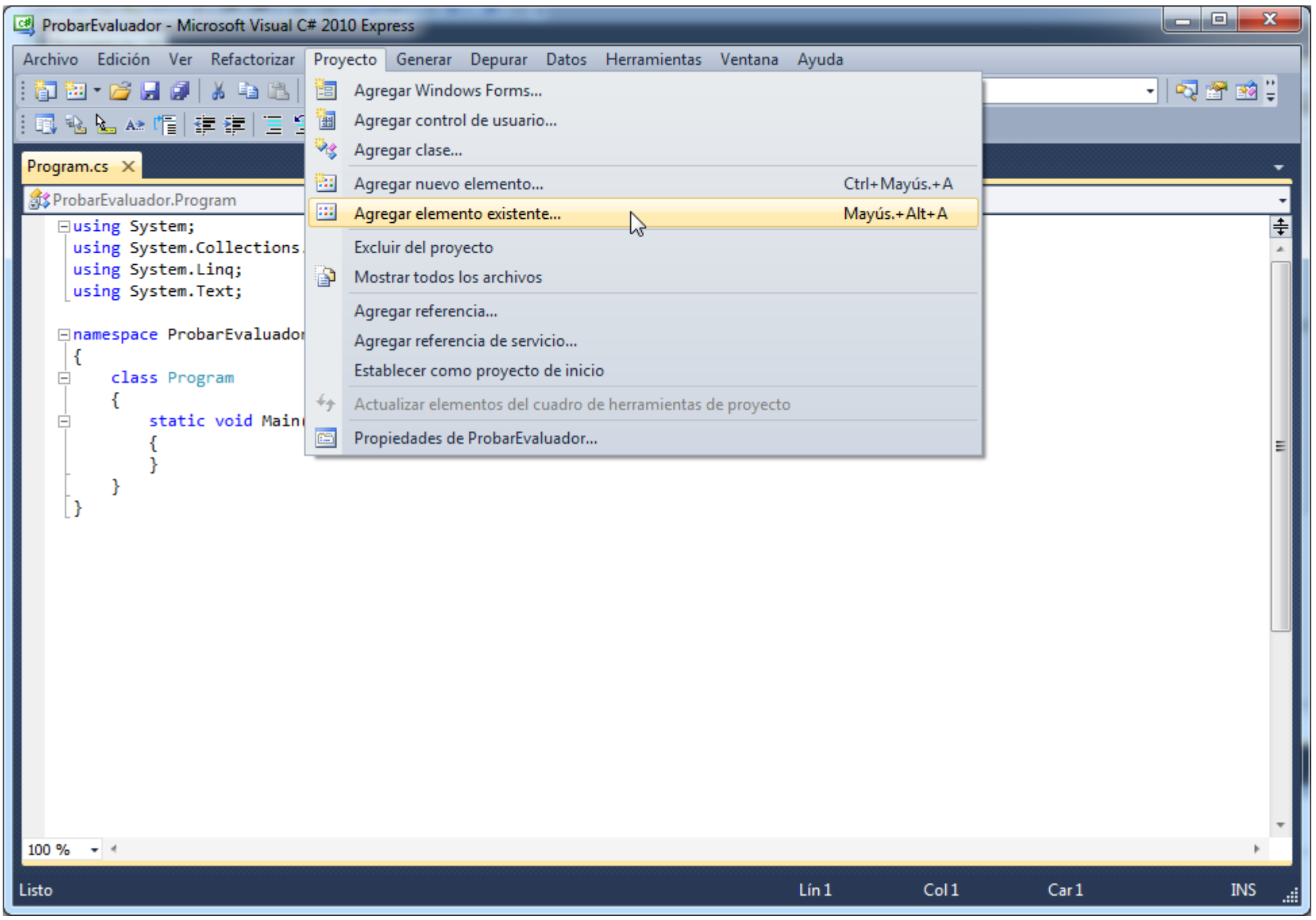


Ilustración 31: Se agrega el código del evaluador de expresiones

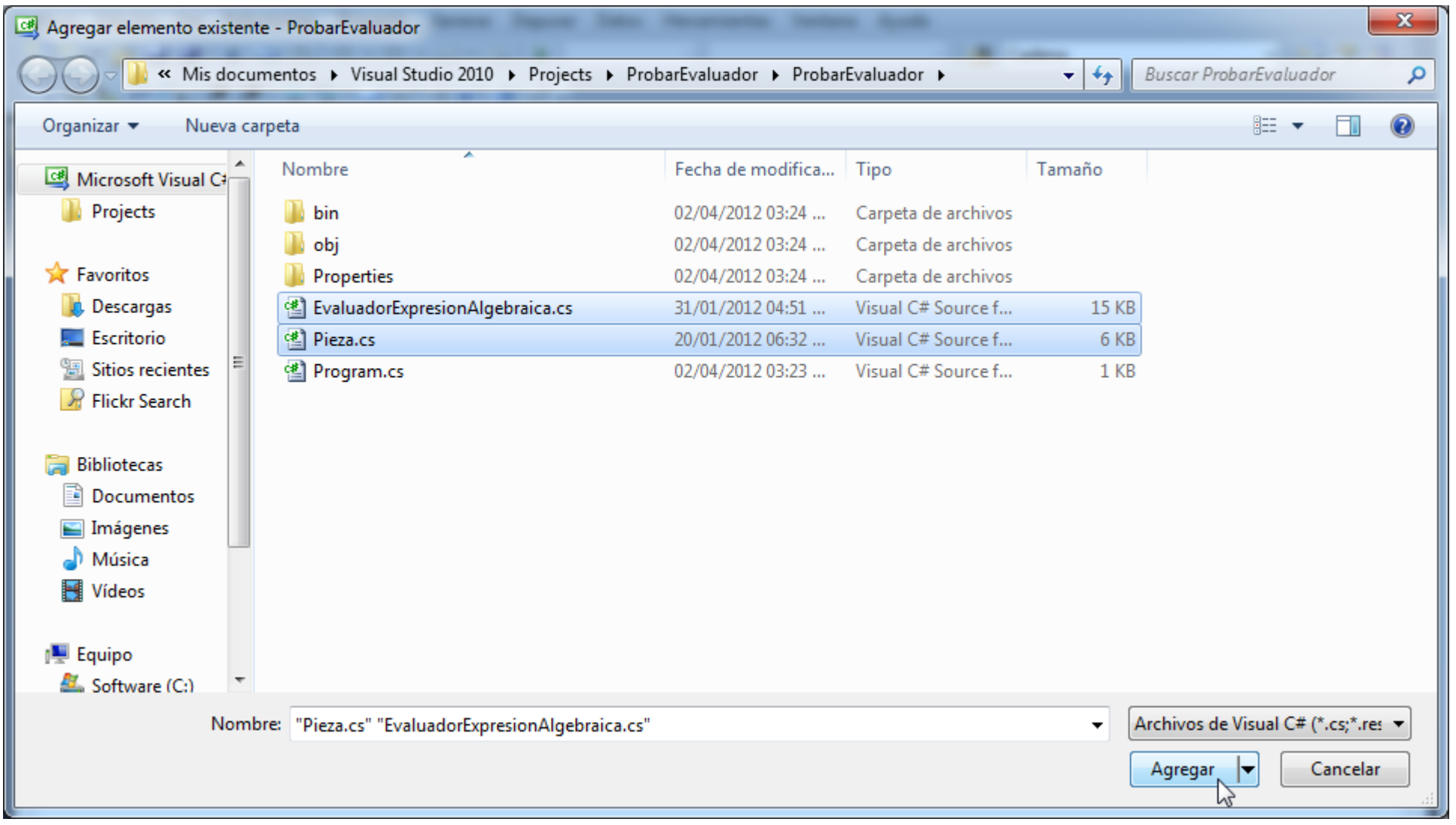


Ilustración 32: Son dos archivos .cs

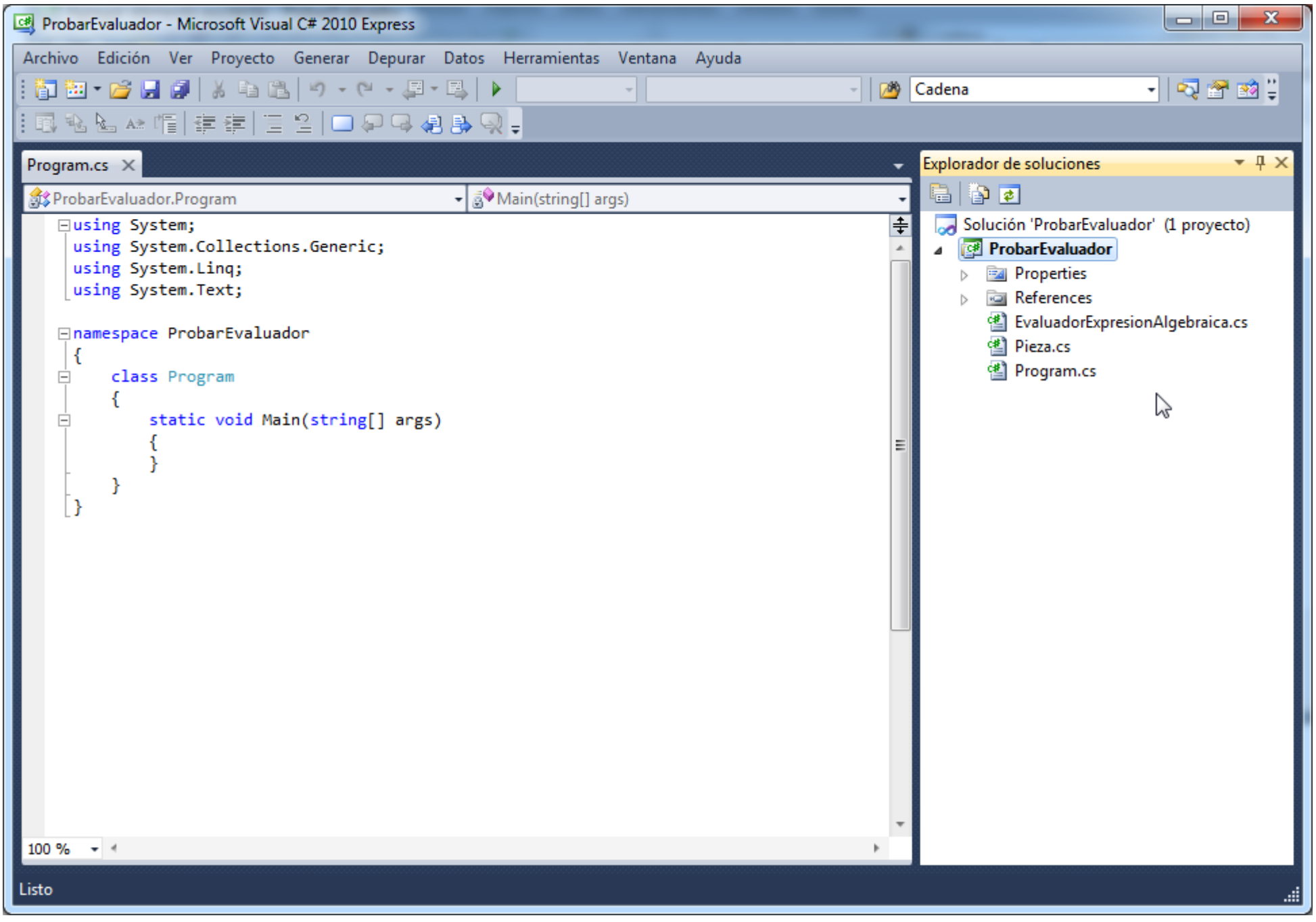


Ilustración 33: Ya están los archivos agregados. Hay que revisar muy bien que coincidan los namespace en los tres archivos .cs

ProbarEvaluador - Microsoft Visual C# 2010 Express

Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Ayuda

Cadena

Pieza.cs EvaluadorExpresionAlgebraica.cs Program.cs

ProbarEvaluador.Program Main(string[] args)

```
/**
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com
 * Página Web: http://darwin.50webs.com
 * Licencia: LGPL
 * Fecha: Enero de 2012
 */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProbarEvaluador
{
    class Program
    {
        static void Main(string[] args)
        {
            EvaluadorExpresionAlgebraica objEvaluador = new EvaluadorExpresionAlgebraica();
            string expresion;
            double valor;

            expresion = "4.763*5.704+(x+sen(3.670/cos(sen(atn(sen(4.877*cos(7.330)))))))";
            objEvaluador.Analizar(expresion);
            objEvaluador.DarValorVariable('x', 13.1729);
            valor = objEvaluador.Evaluar();
            Console.WriteLine(valor);
            Console.ReadKey();
        }
    }
}
```

Explorador de soluciones

Solución 'ProbarEvaluador' (1 proye

ProbarEvaluador

Properties

References

EvaluadorExpresionAlgebraic

Pieza.cs

Program.cs

100 %

Listo Lín 23 Col 89 Car 89 INS

Ilustración 34: Código de prueba del evaluador de expresiones

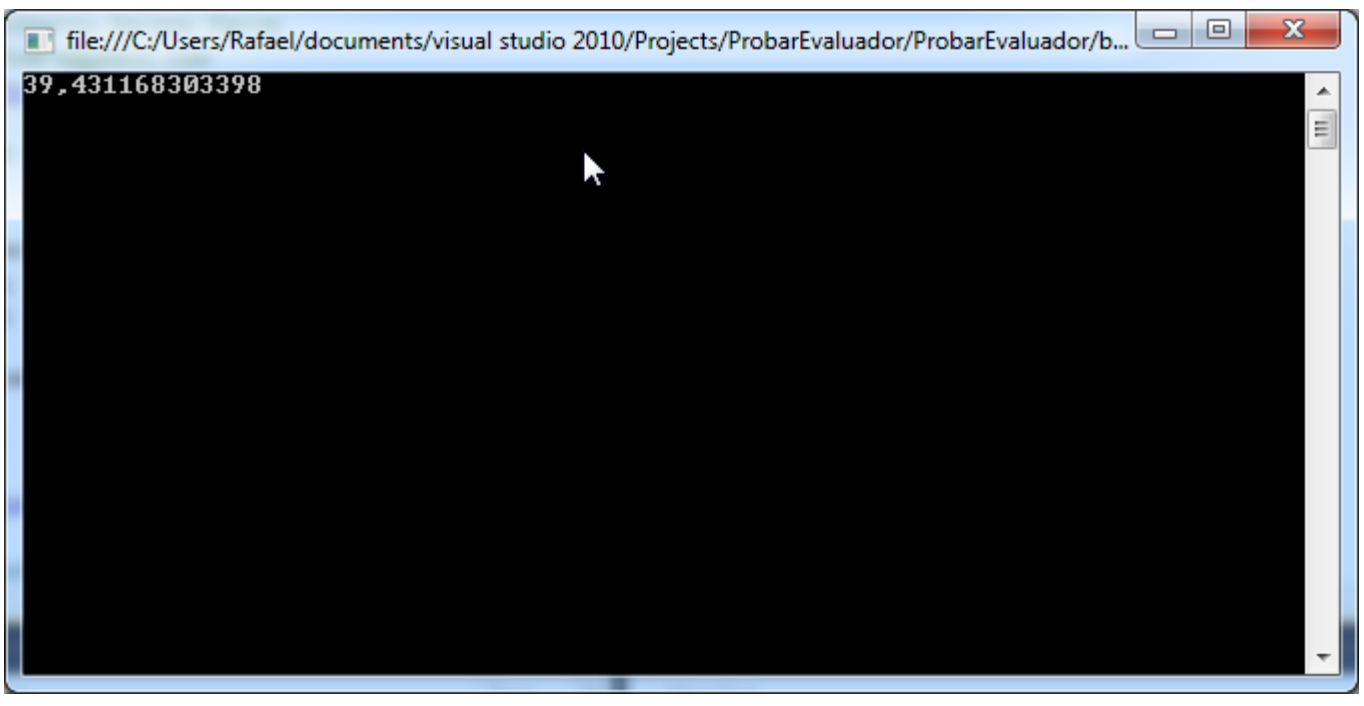


Ilustración 35: Ejemplo de ejecución

Visual Basic .NET

Puede hacer uso de Microsoft Visual Basic .Net Express 2010 (es gratuita). En: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-basic-express

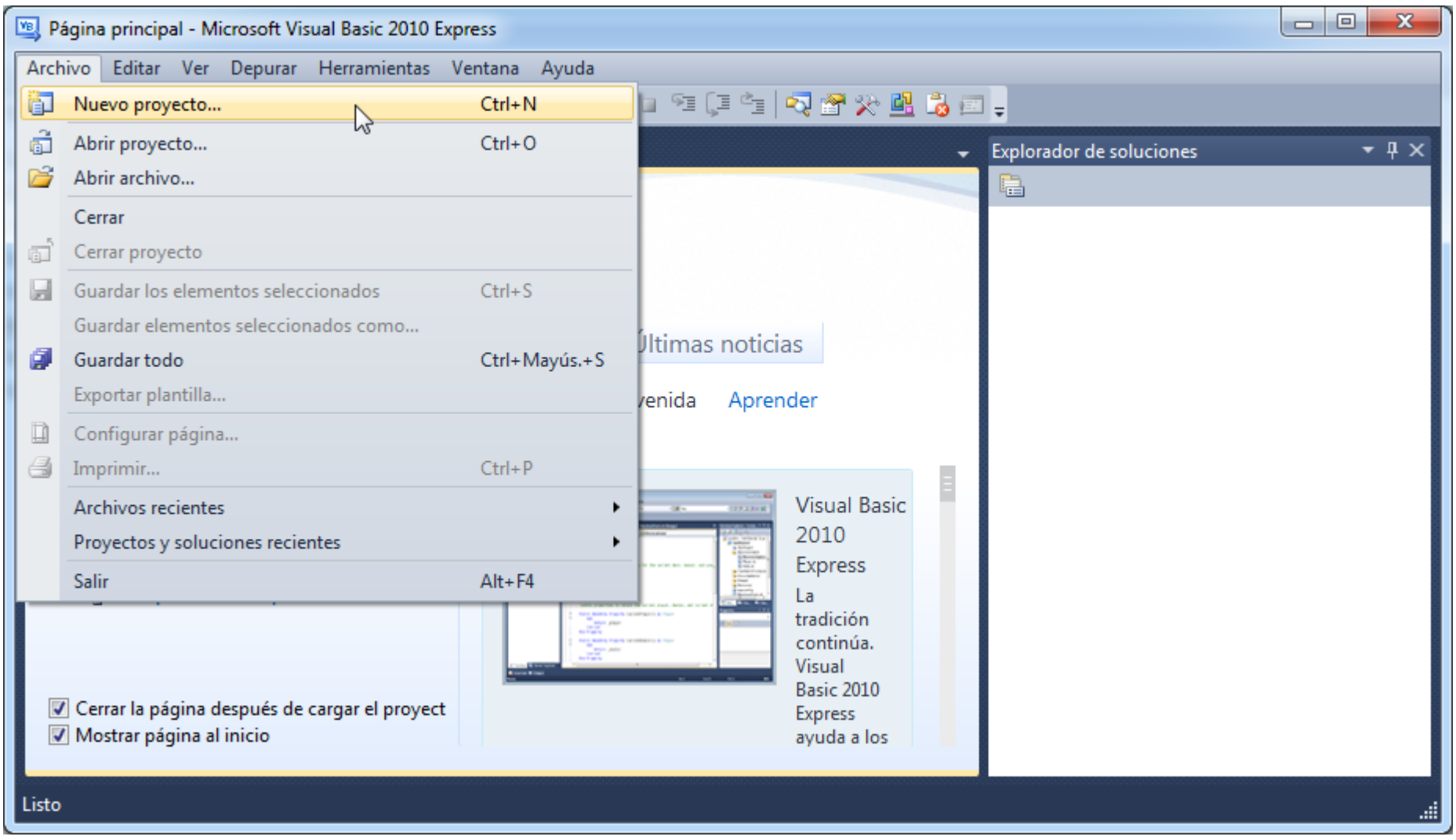


Ilustración 36: Nuevo proyecto en Visual Basic 2010

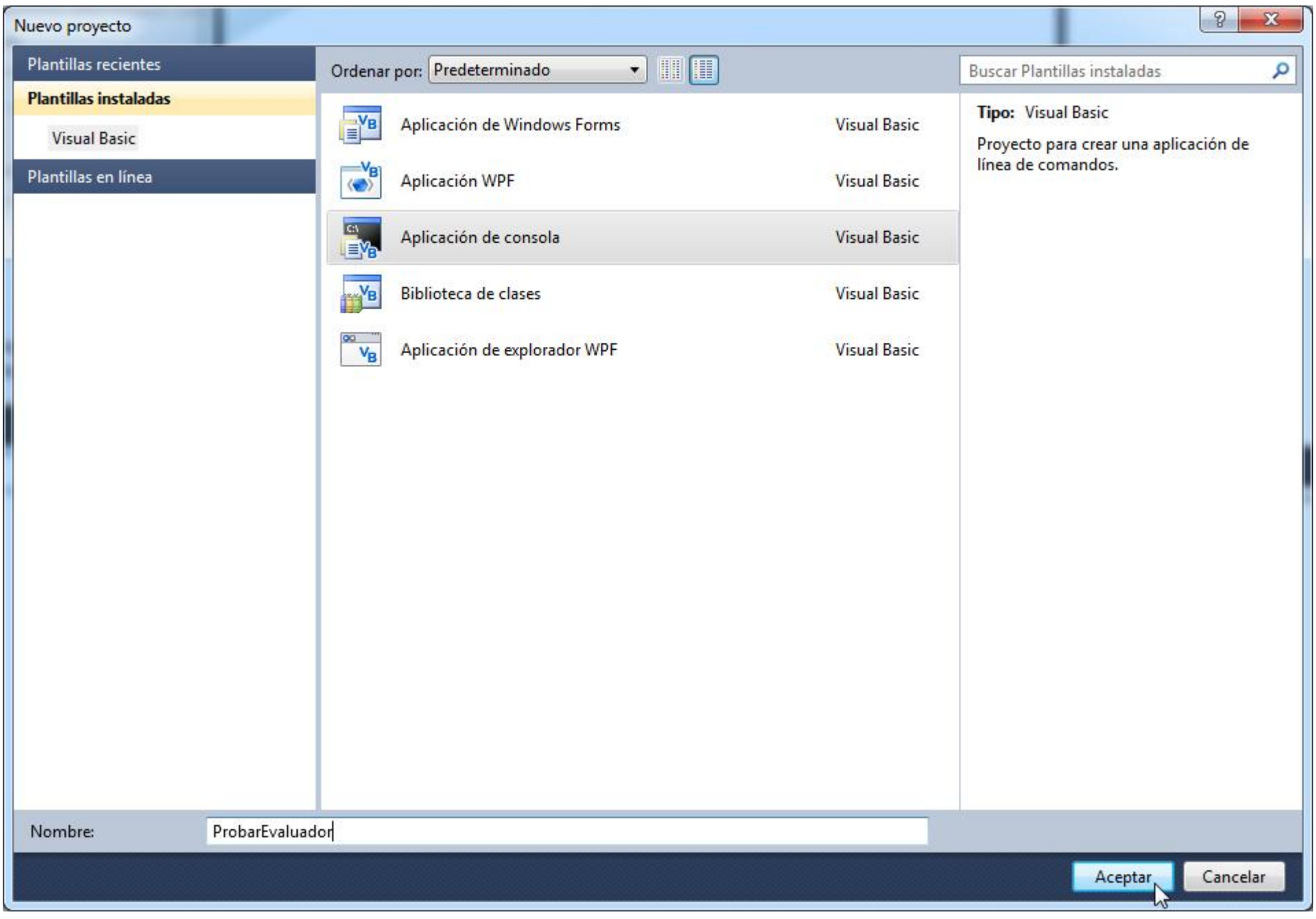


Ilustración 37: Aplicación de consola

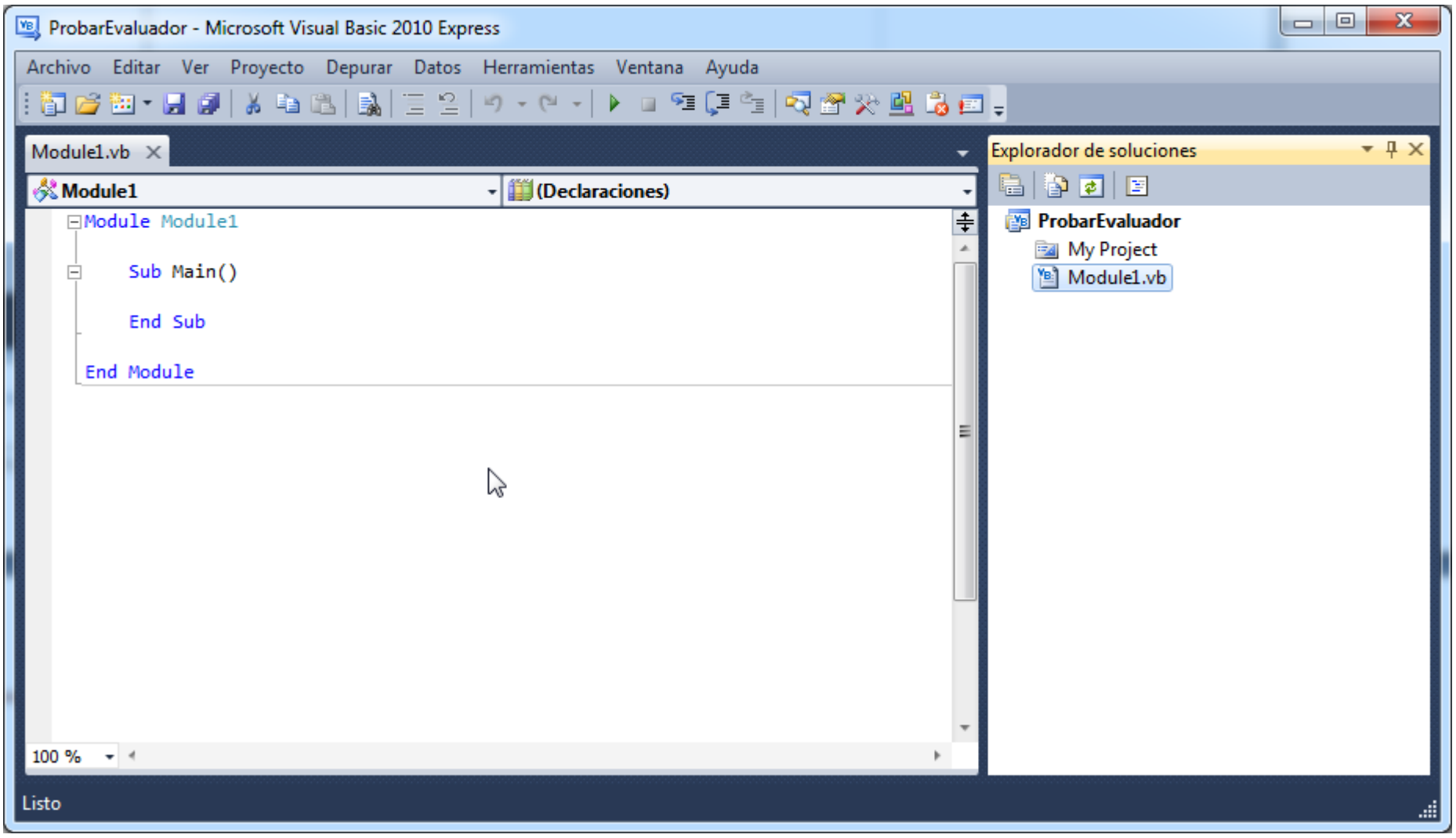


Ilustración 38: Código por defecto

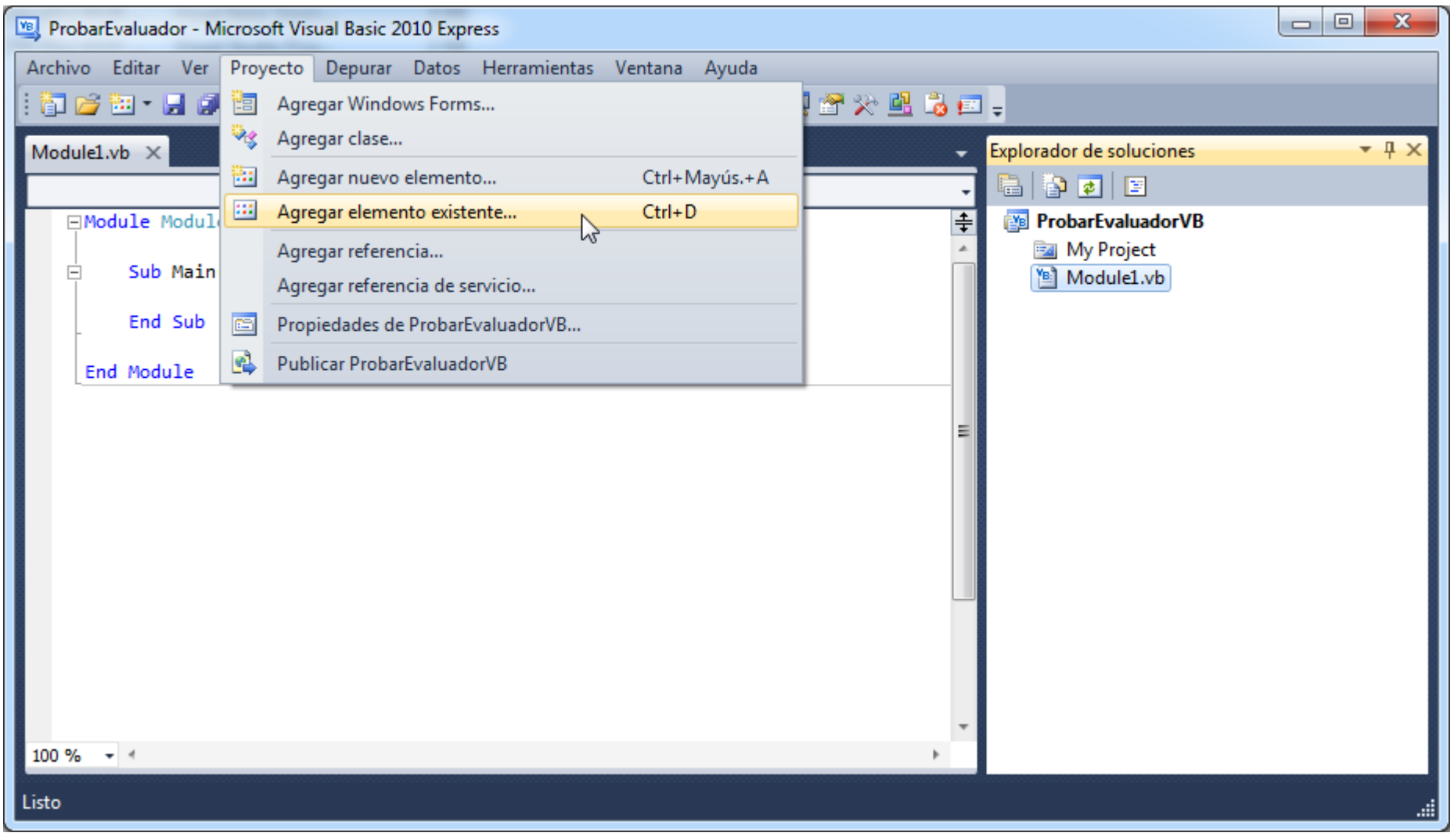


Ilustración 39: Agregar el código del evaluador de expresiones

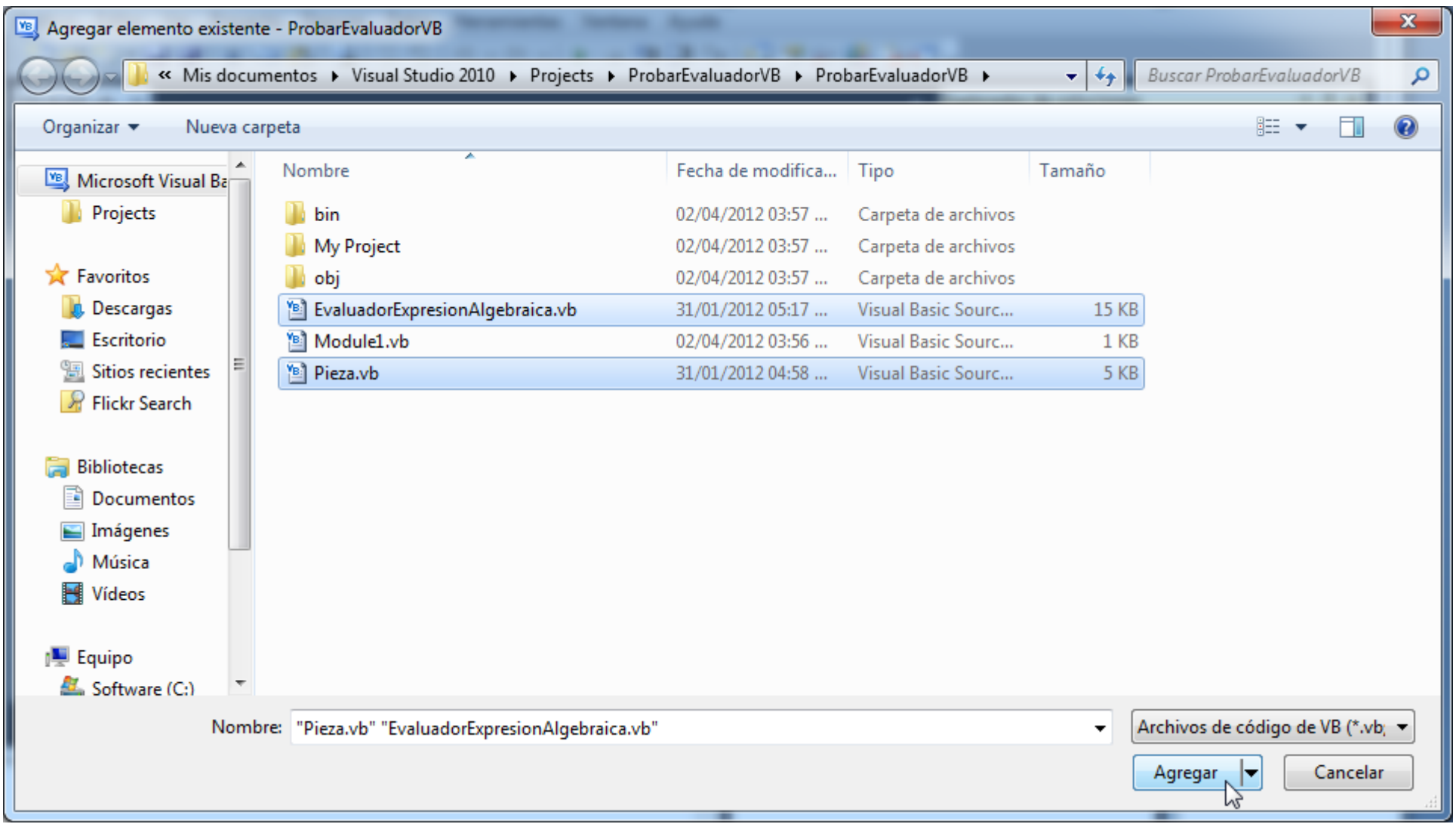


Ilustración 40: Dos archivos requiere el evaluador de expresiones

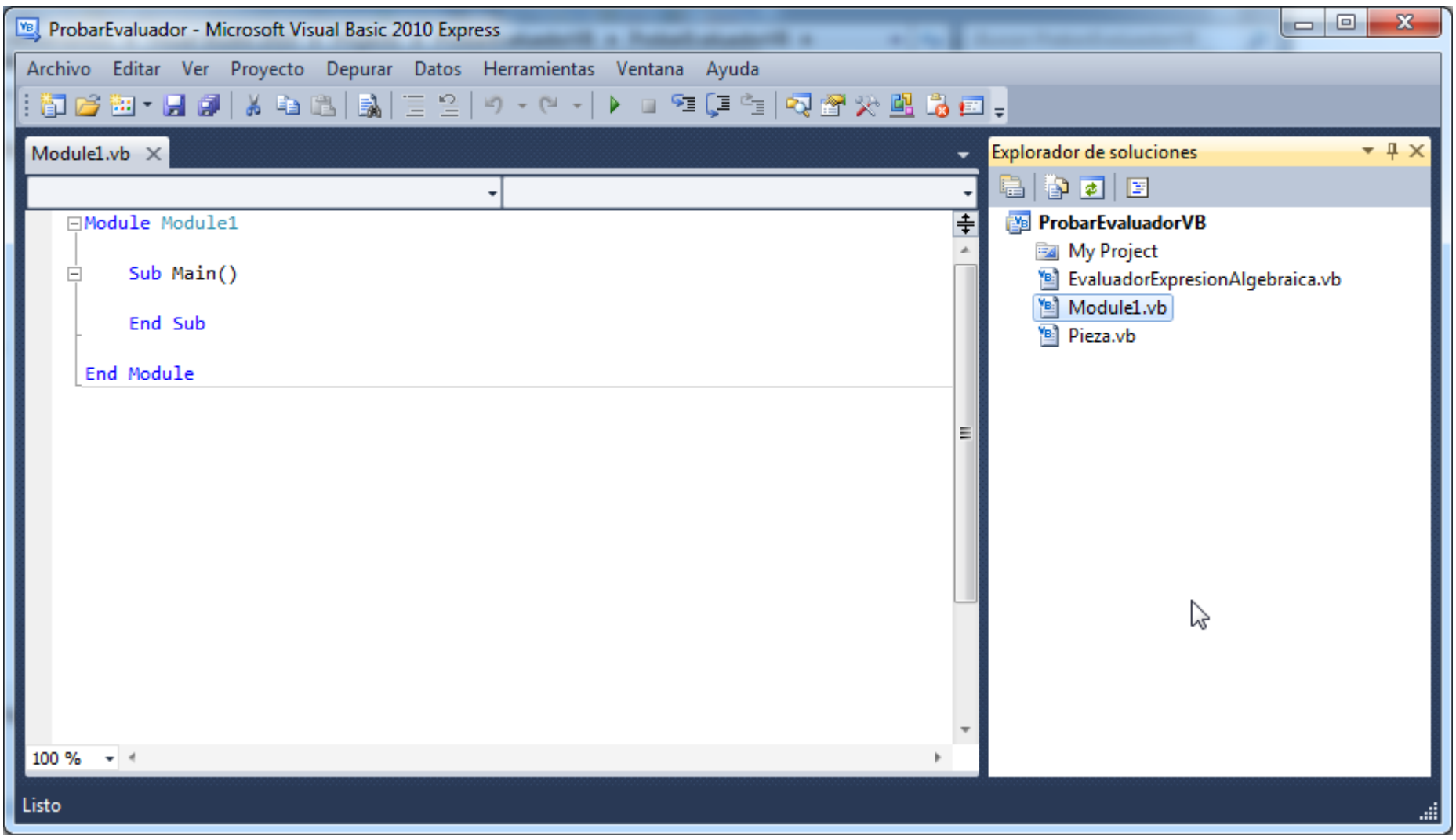


Ilustración 41: Evaluador agregado al proyecto

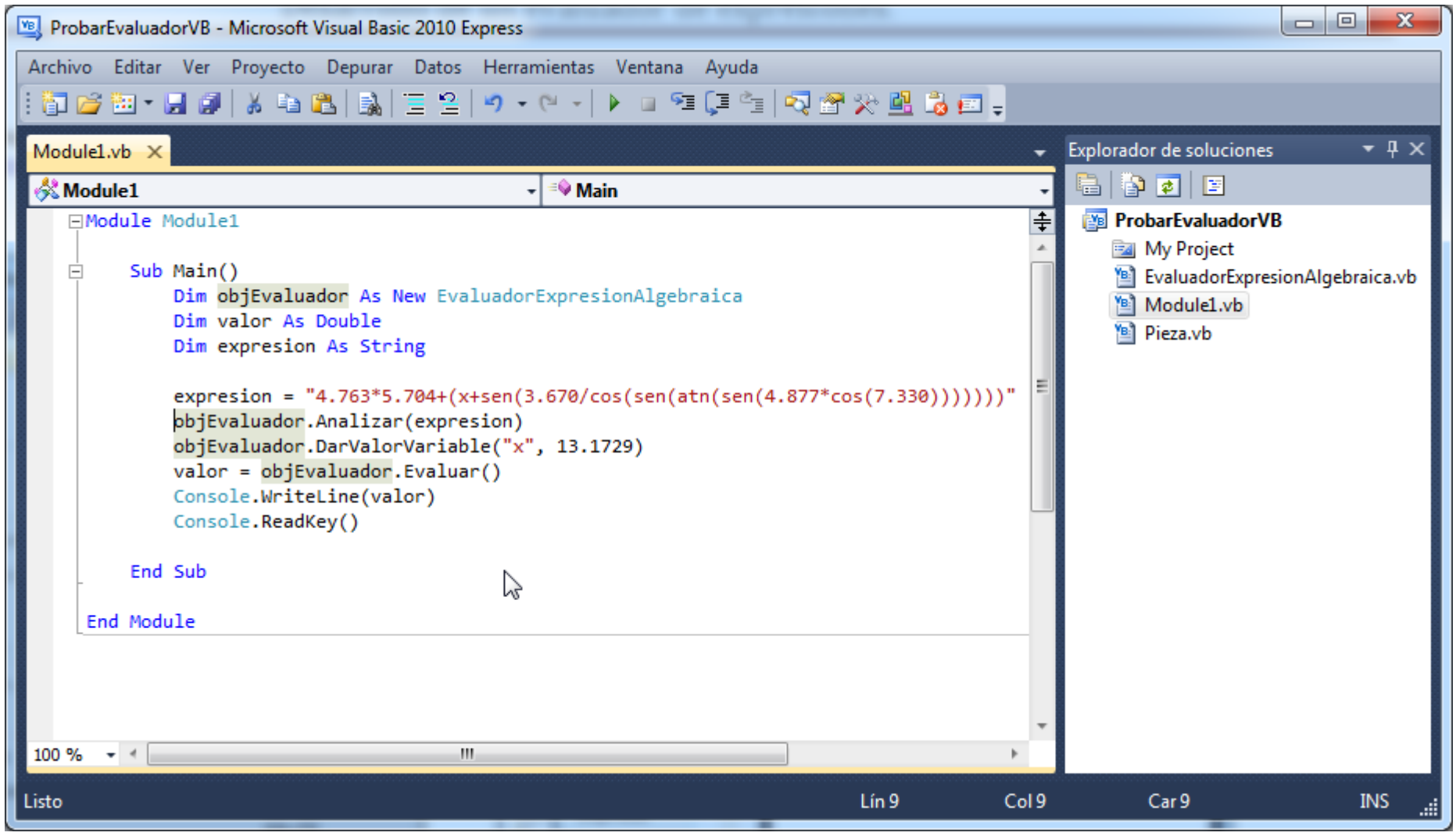


Ilustración 42: Código para probar el evaluador de expresiones

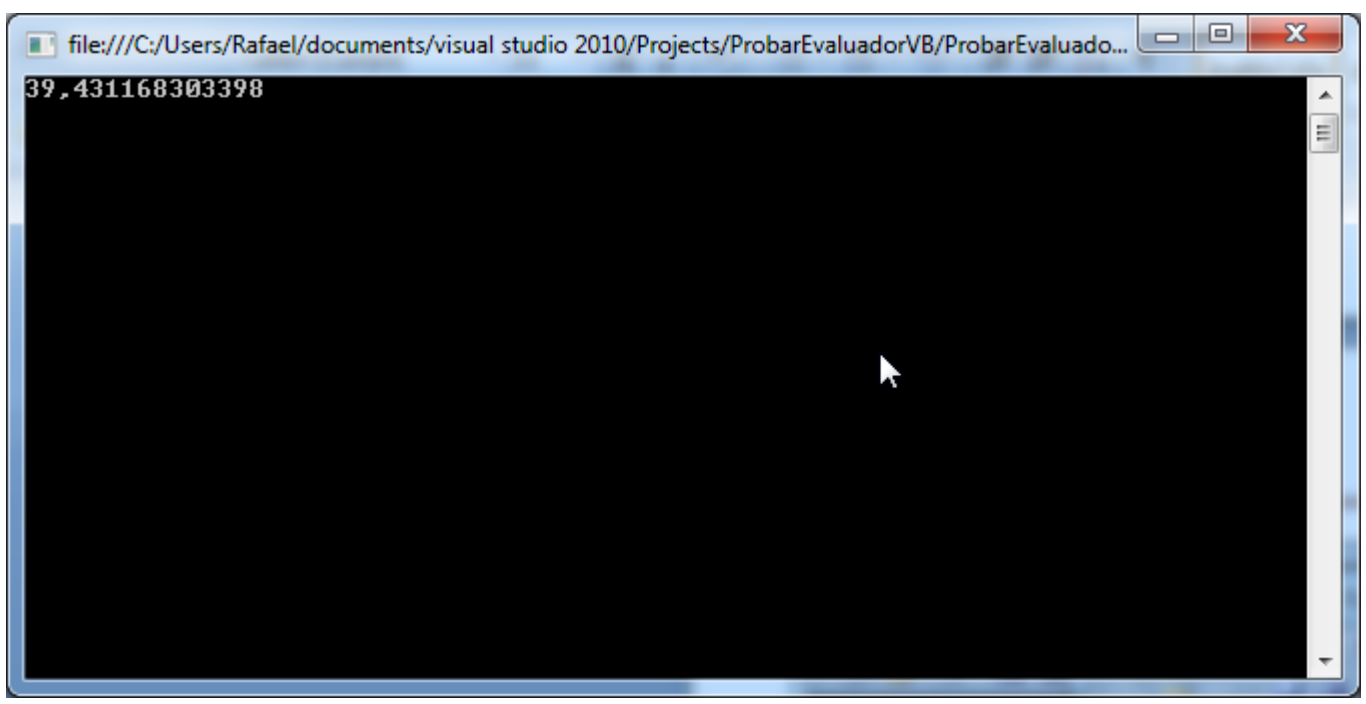


Ilustración 43: Ejecución de prueba

Java

Puede hacer uso de NetBeans en: http://netbeans.org/

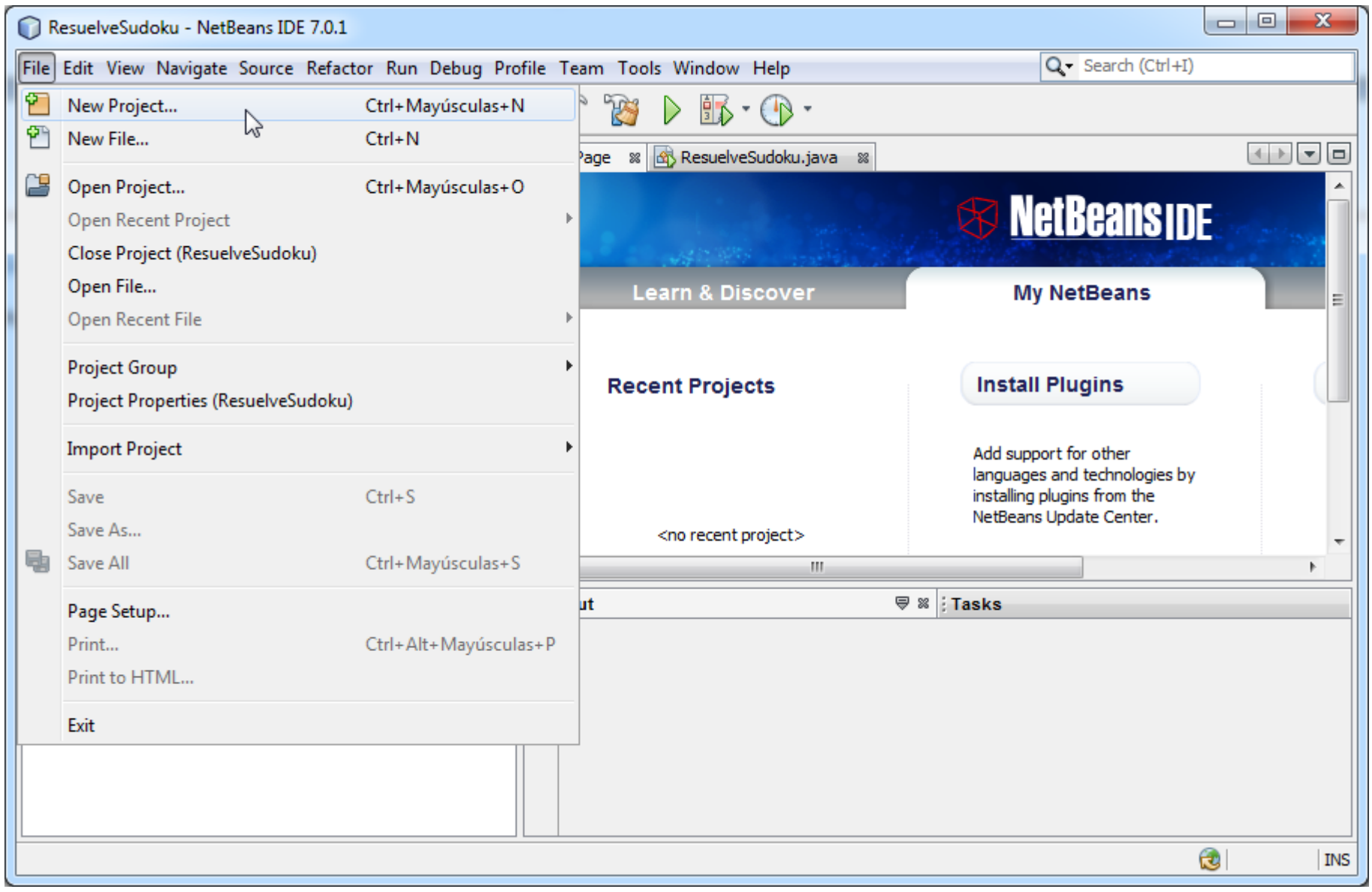


Ilustración 44: Nuevo proyecto en NetBeans

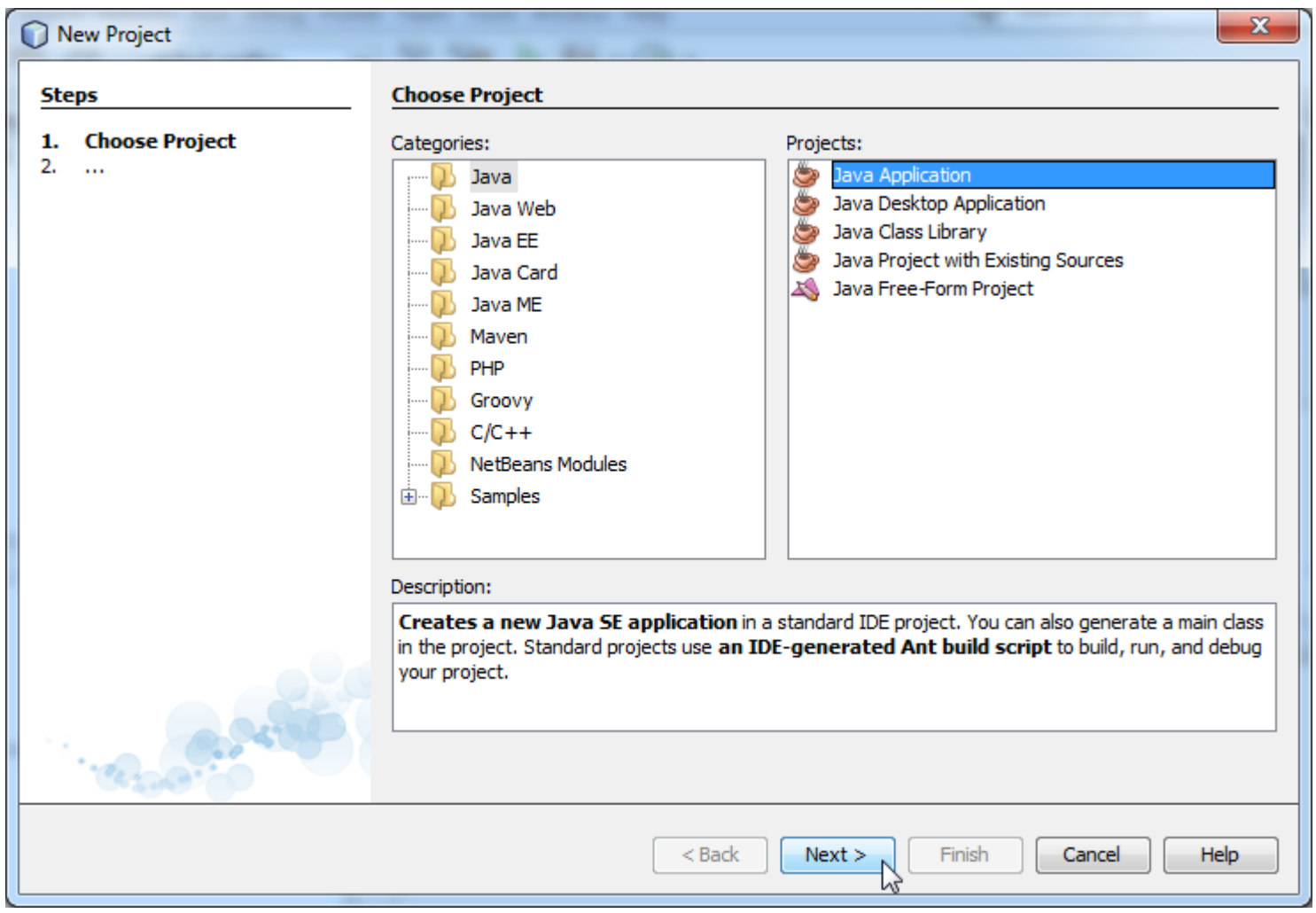


Ilustración 45: Una aplicación en Java

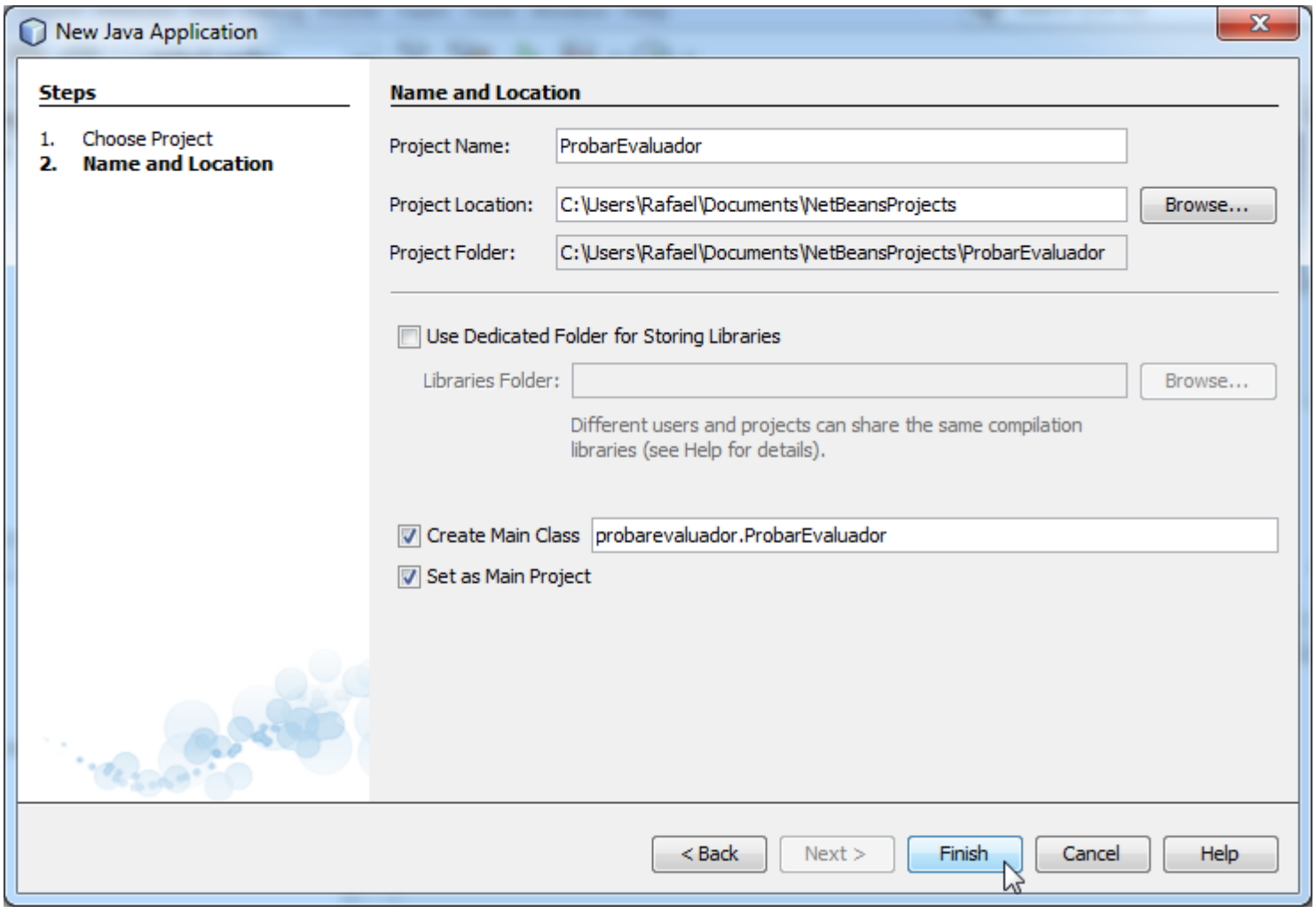


Ilustración 46: Características del proyecto en Java

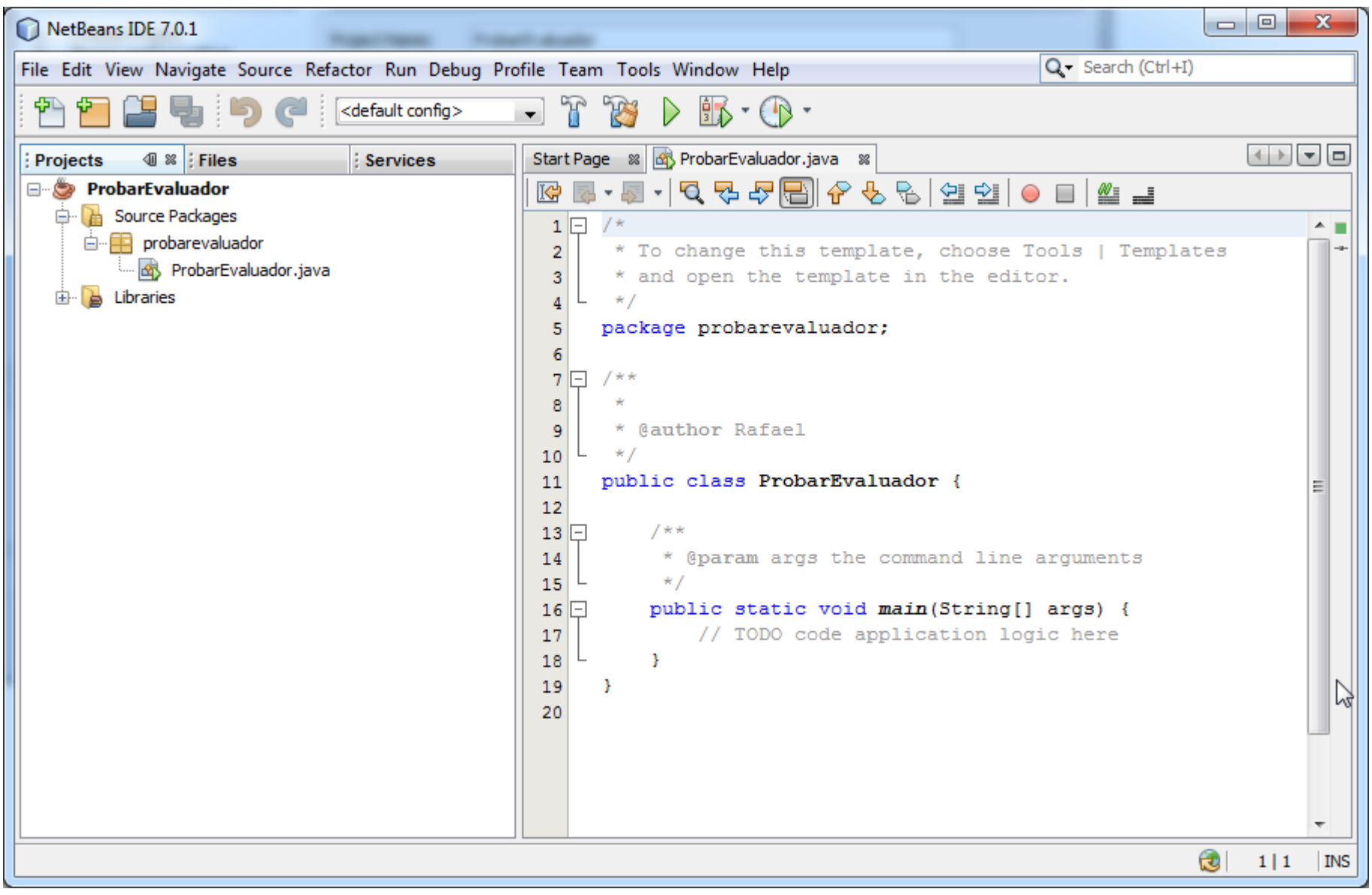


Ilustración 47: Código por defecto en Java

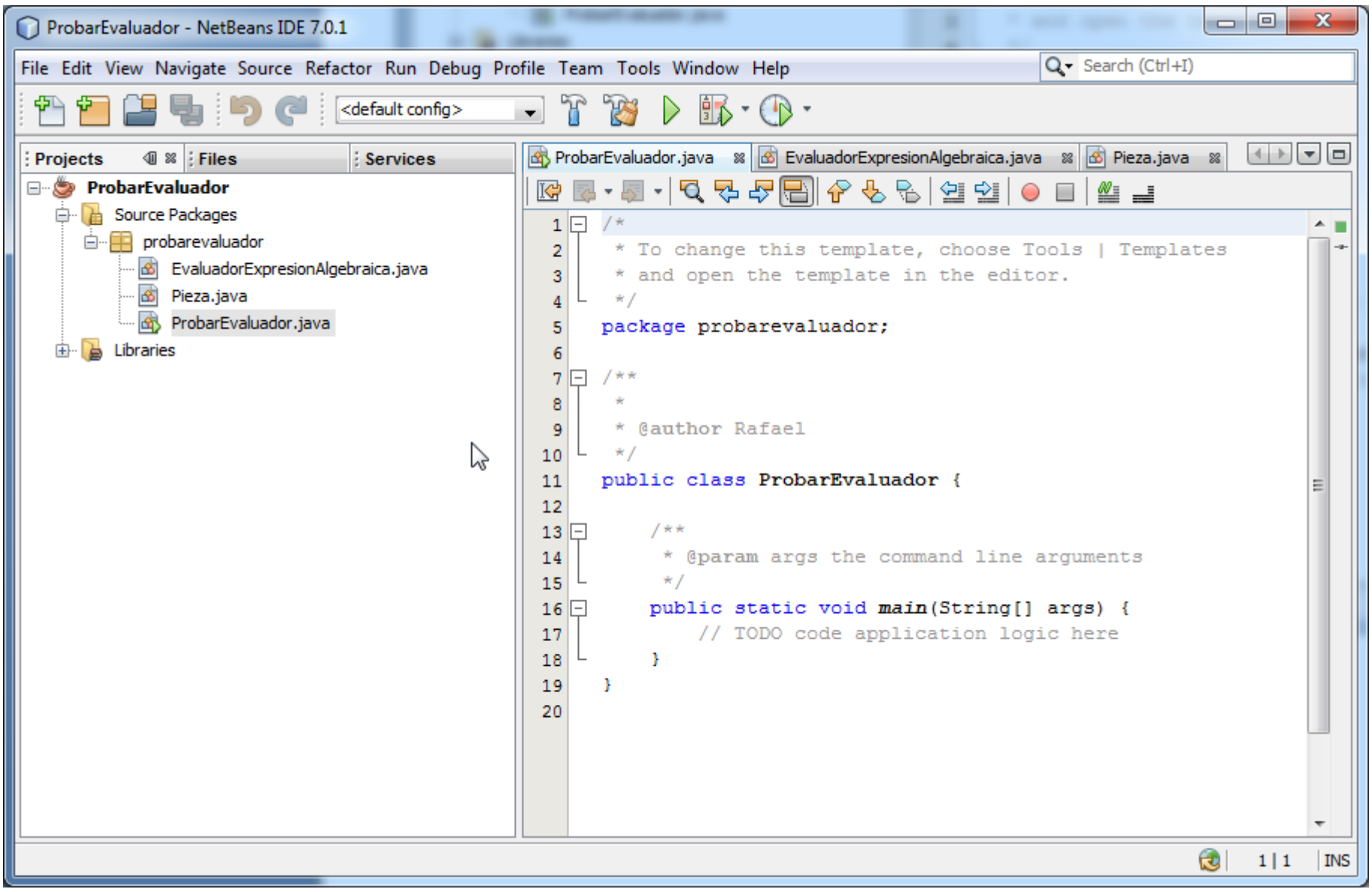


Ilustración 48: Se agregan los archivos necesarios del evaluador de expresiones

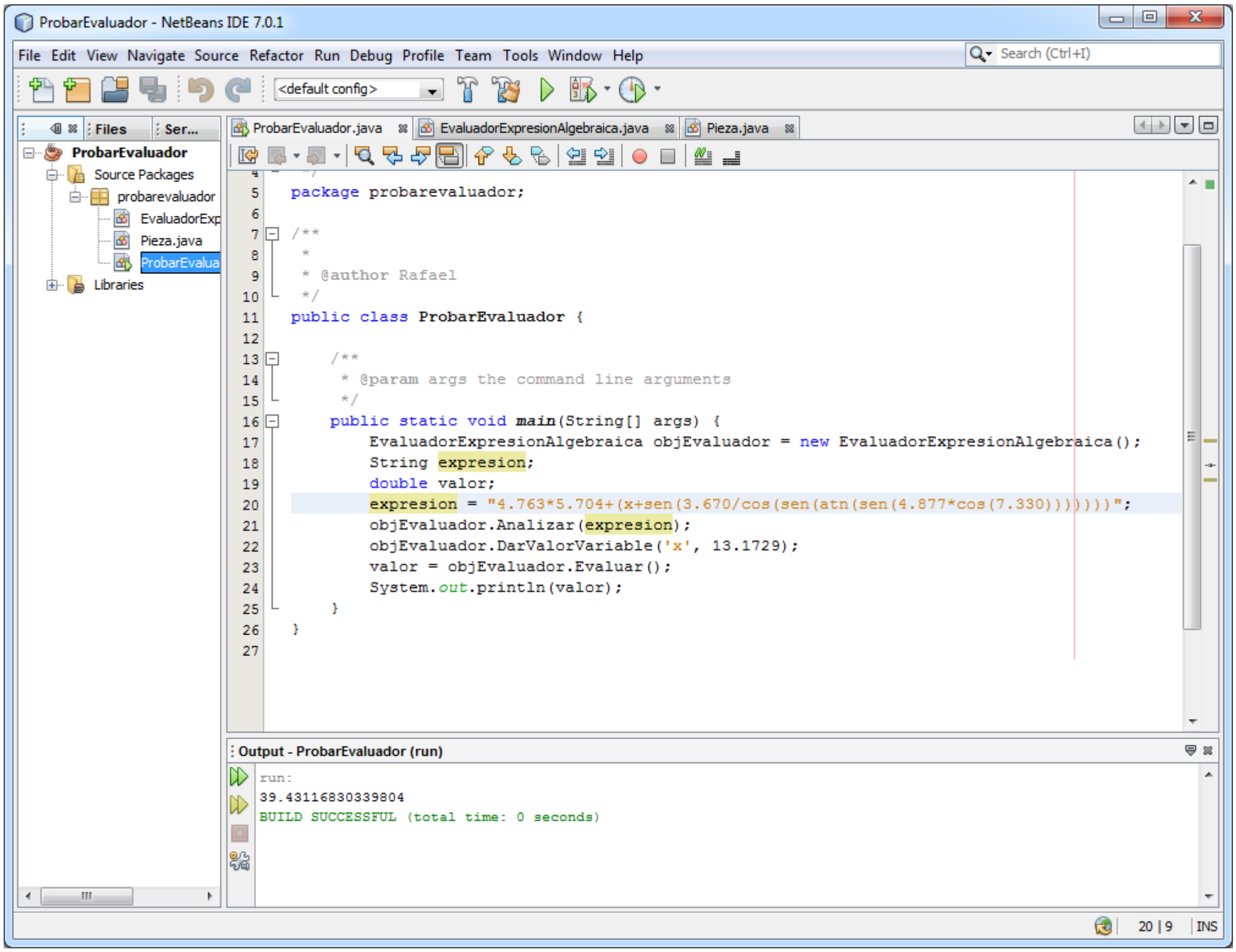


Ilustración 49: Código para probar el evaluador y ejecución

PHP

Puede hacer uso de un editor como Notepad++ en: http://notepad-plus-plus.org/ , ún navegador moderno y un entorno de ejecución de programas en WEB como WampServer http://www.wampserver.com/en/

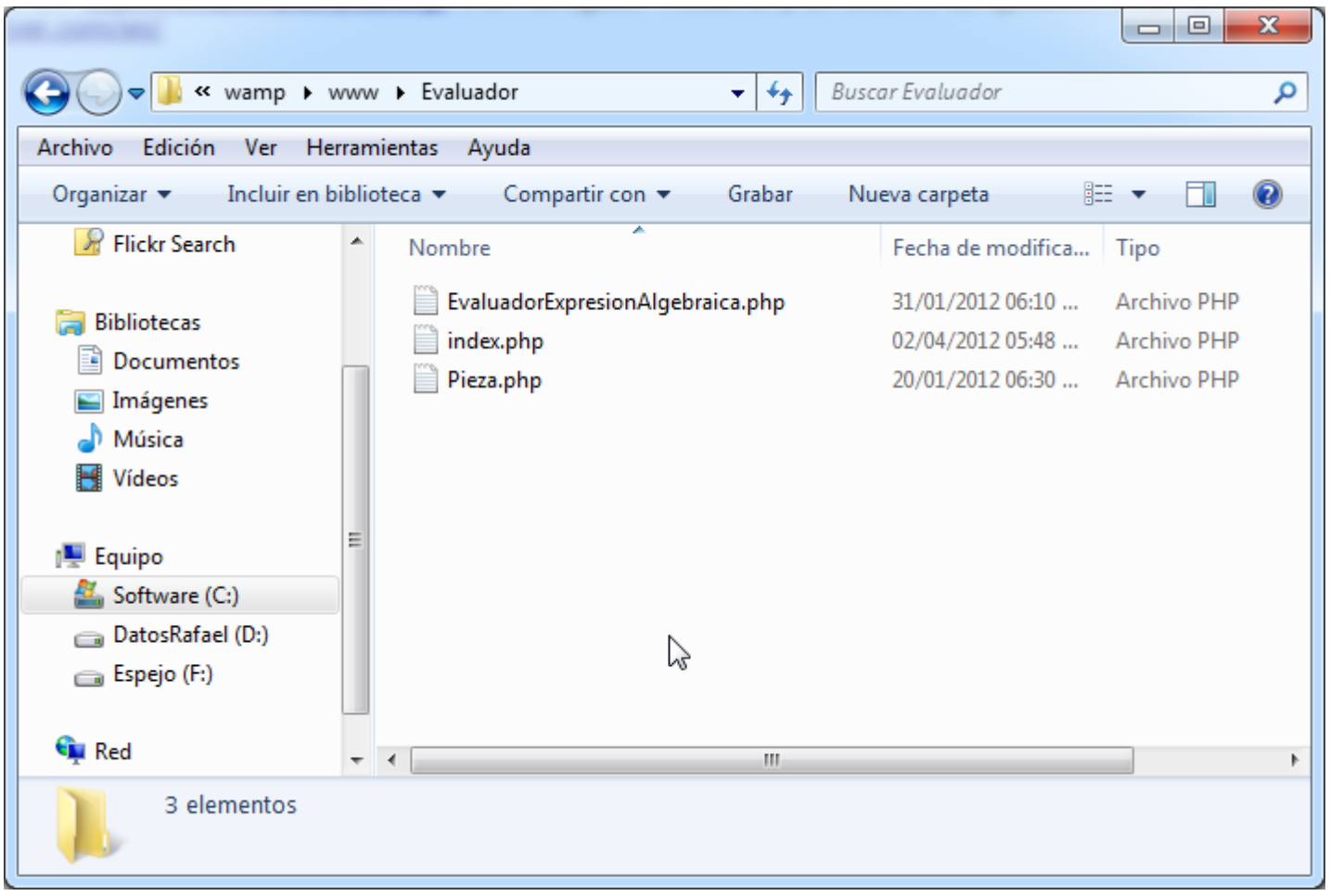


Ilustración 50: Los archivos en la ubicación para ser ejecutados por wampserver (por defecto es en c:\wamp\www)

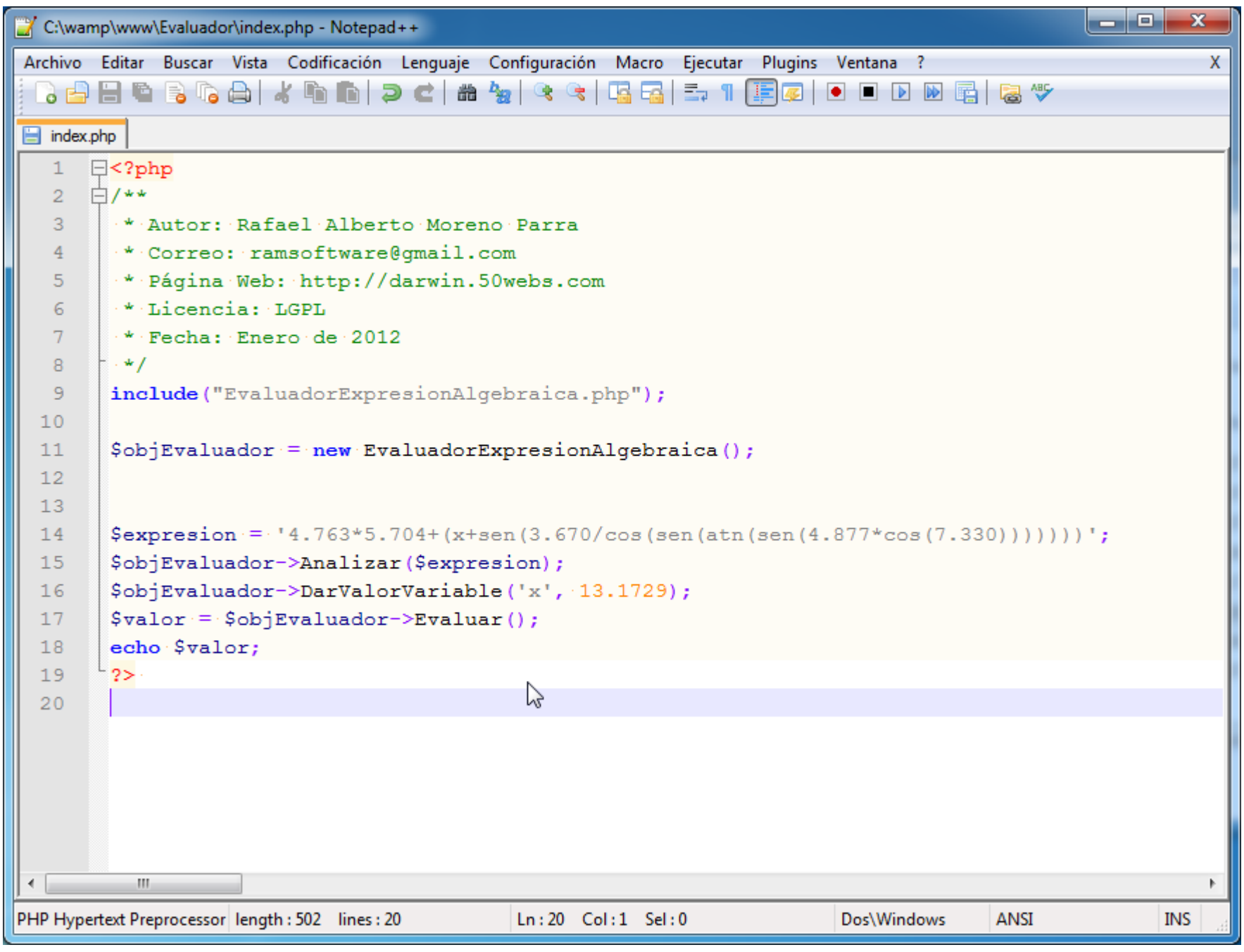


Ilustración 51: Código para probar el evaluador

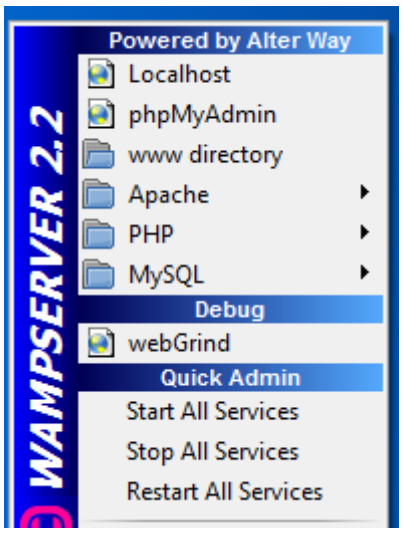


Ilustración 52: Wampserver ejecutándose

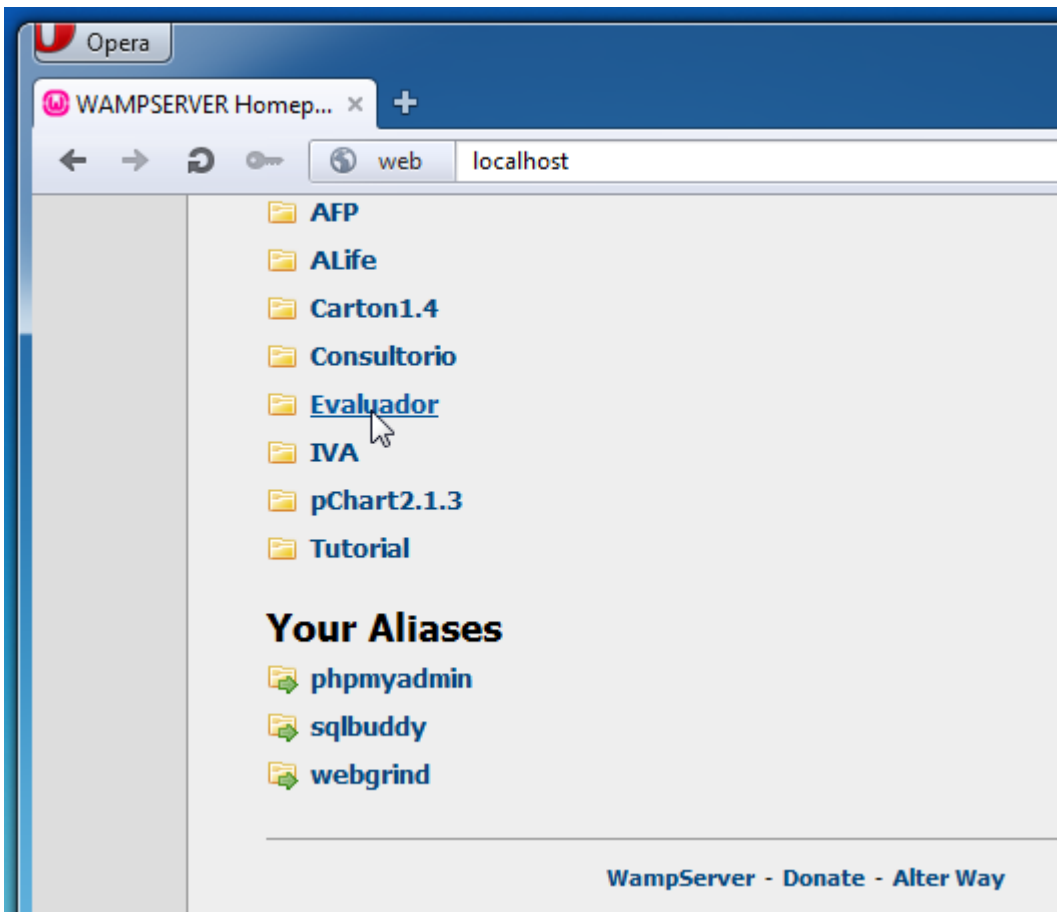


Ilustración 53: El proyecto del evaluador de expresiones

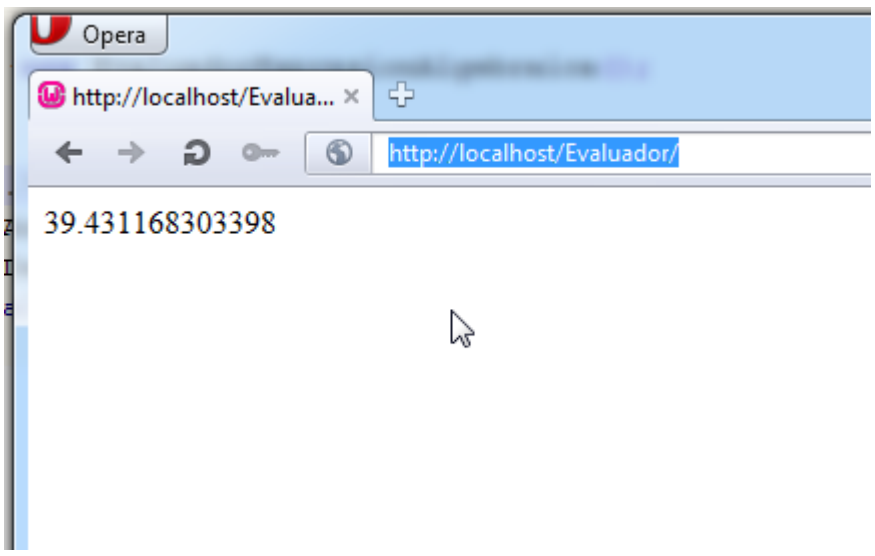


Ilustración 54: Ejecución del evaluador

JavaScript

Puede hacer uso de un editor como Notepad++ en: http://notepad-plus-plus.org/ y de algún navegador moderno para ejecutar JavaScript

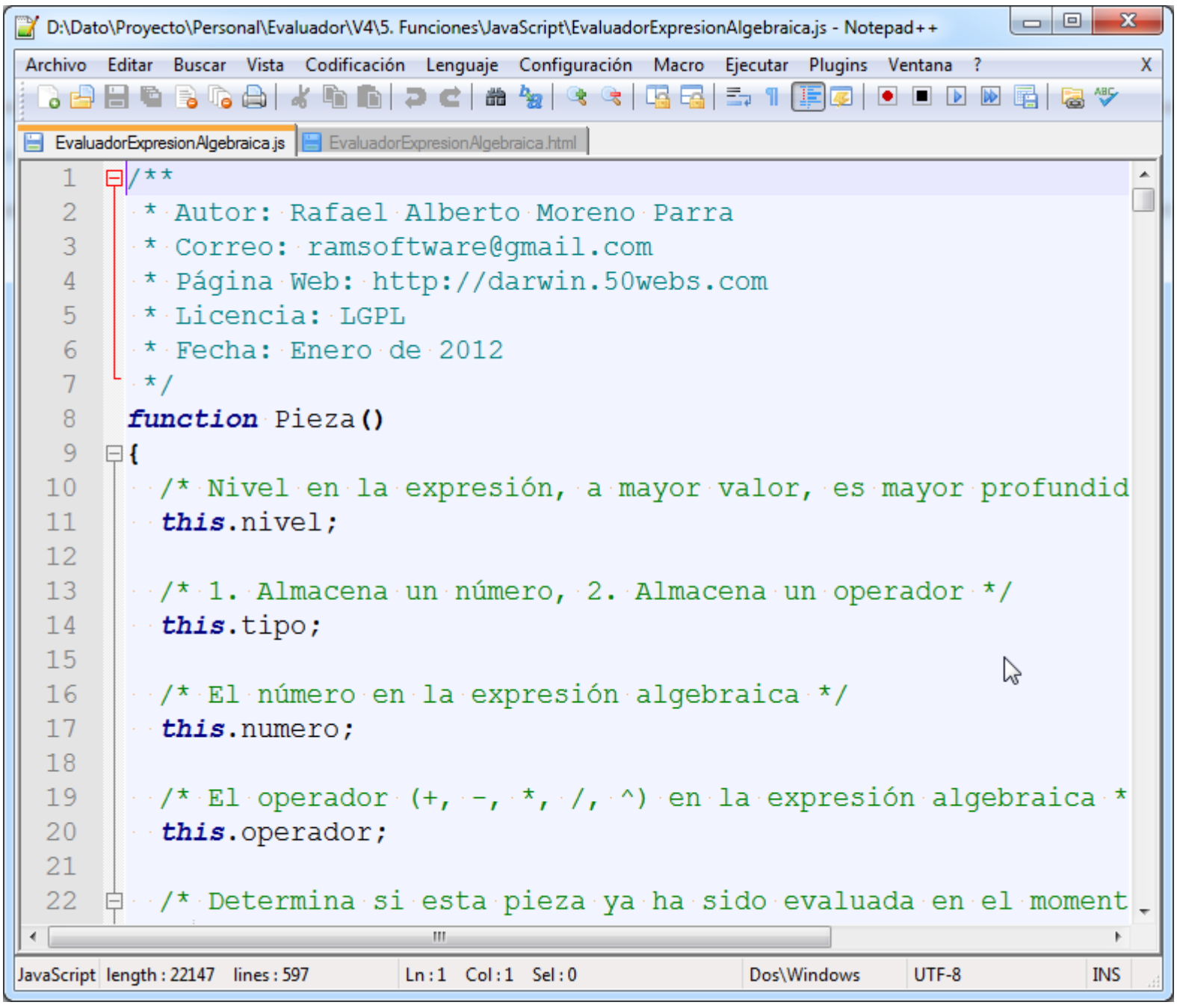


Ilustración 55: El evaluador en su archivo Javascript .js

C:\Users\Rafael\Desktop\PruebaEvaluador\EvaluadorExpresionAlgebraica.html - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Macro Ejecutar Plugins Ventana ? X

EvaluadorExpresionAlgebraica.html | EvaluadorExpresionAlgebraica.js

```
1  <html>
2    <body>
3      <script type="text/javascript" src="EvaluadorExpresionAlgebraica.js"> </script>
4      <script type="text/javascript">
5        /* Probando el evaluador de expresiones */
6        objEvaluador = new EvaluadorExpresionAlgebraica();
7        expresion = "4.763*5.704+(x+sen(3.670/cos(sen(atn(sen(4.877*cos(7.330))))))))";
8        objEvaluador.Analizar(expresion);
9        objEvaluador.DarValorVariable('x', 13.1729);
10       valor = objEvaluador.Evaluar();
11       document.write(valor);
12       </script>
13   </body>
14 </html>
15
```

Hyper Text Markup Languag | length : 535 lines : 15 | Ln : 7 Col : 84 Sel : 0 | Dos\Windows | UTF-8 | INS

Ilustración 56: El archivo HTML que invoca el programa en JavaScript

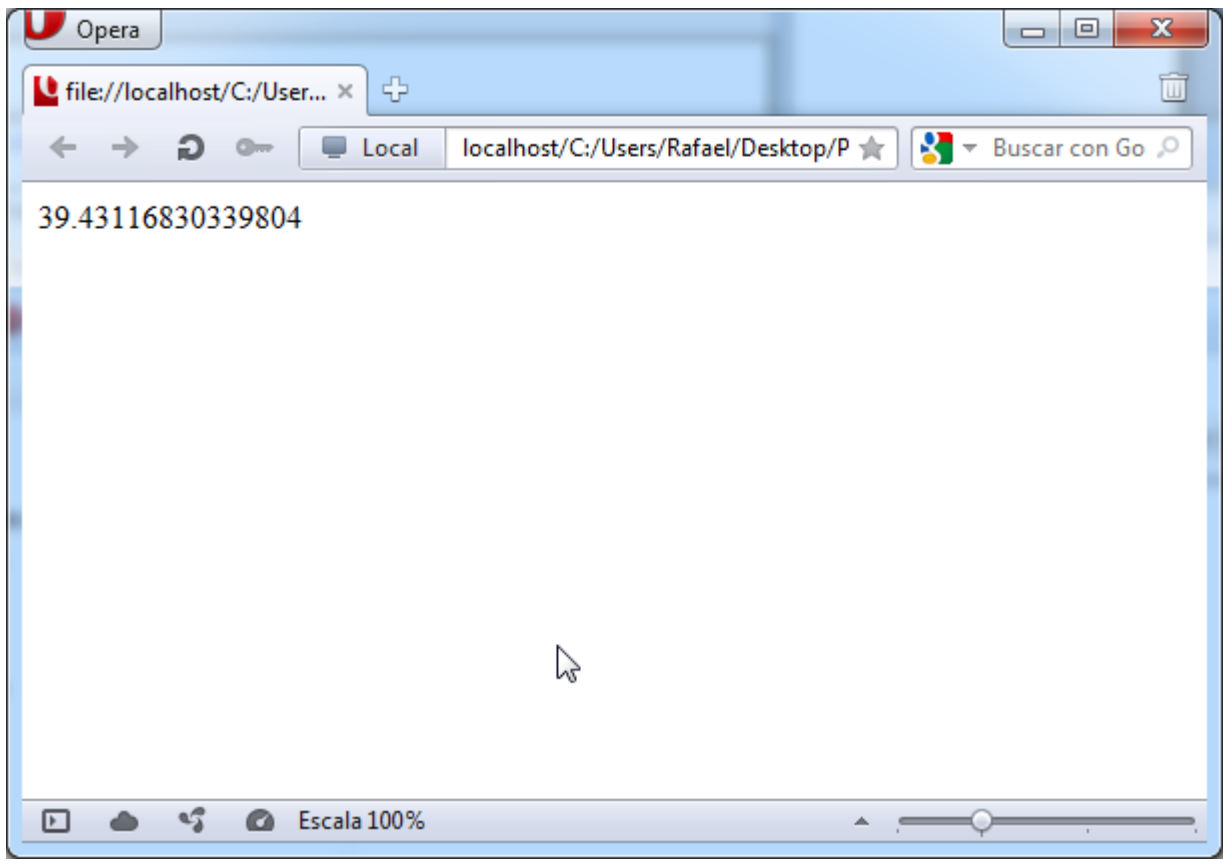


Ilustración 57: Ejecución directa en un navegador

Object Pascal

Puede hacer uso del IDE libre Lazarus en http://sourceforge.net/projects/lazarus/?_test=b o el propietario Embarcadero Delphi XE2 en http://www.embarcadero.com/products/delphi

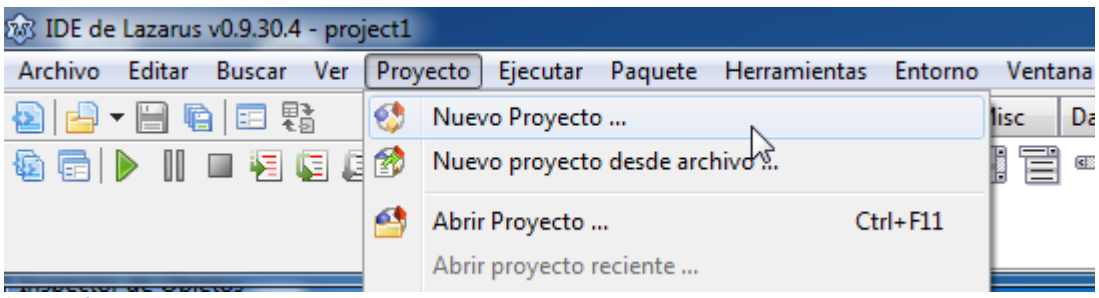


Ilustración 58: Se crea un nuevo prooyecto en Lazarus

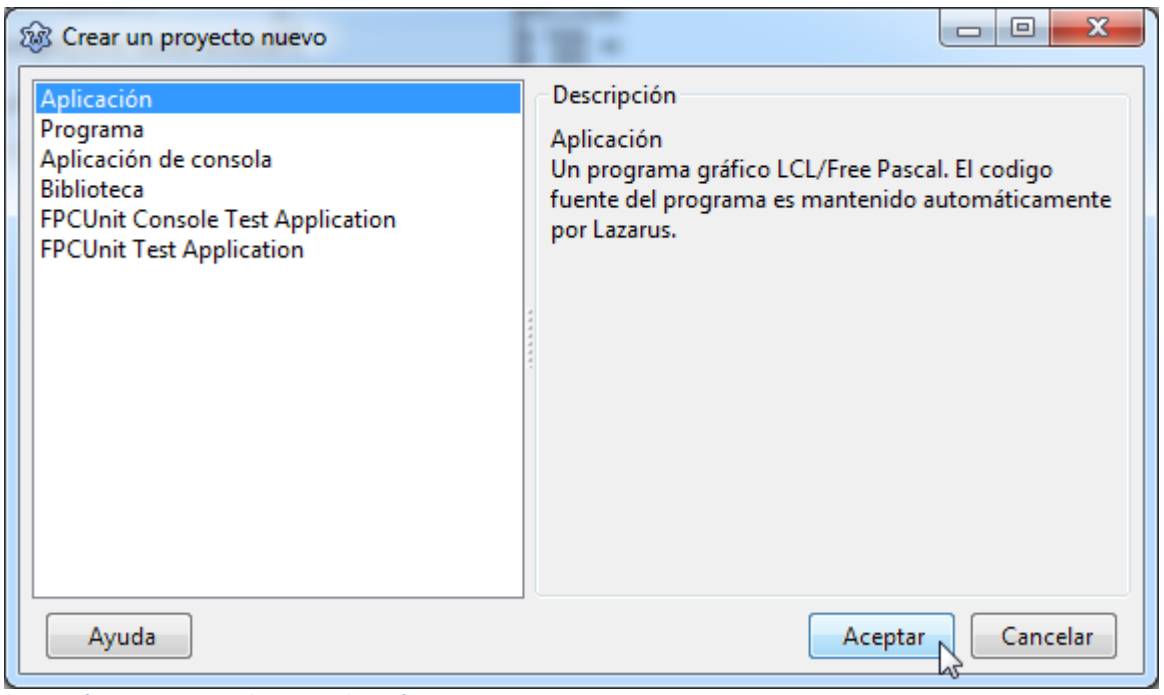


Ilustración 59: Prouecto de tipo Aplicación

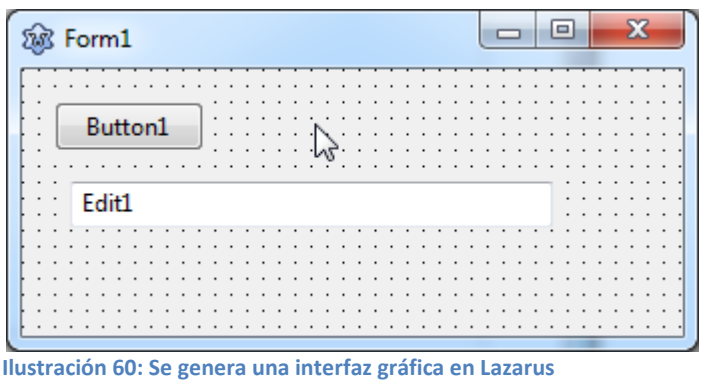


Ilustración 60: Se genera una interfaz gráfica en Lazarus

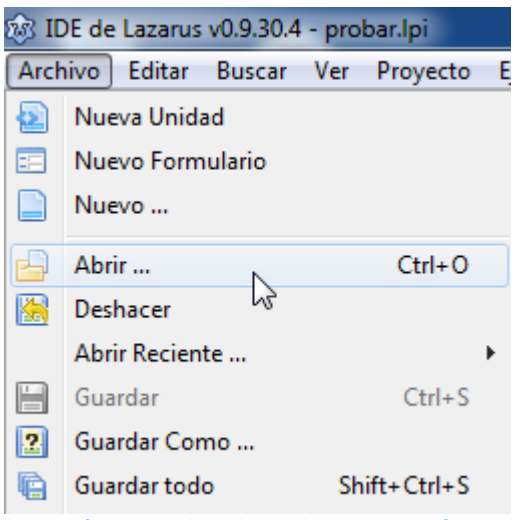


Ilustración 61: Se abren los archivos que conforman el evaluador de expresiones

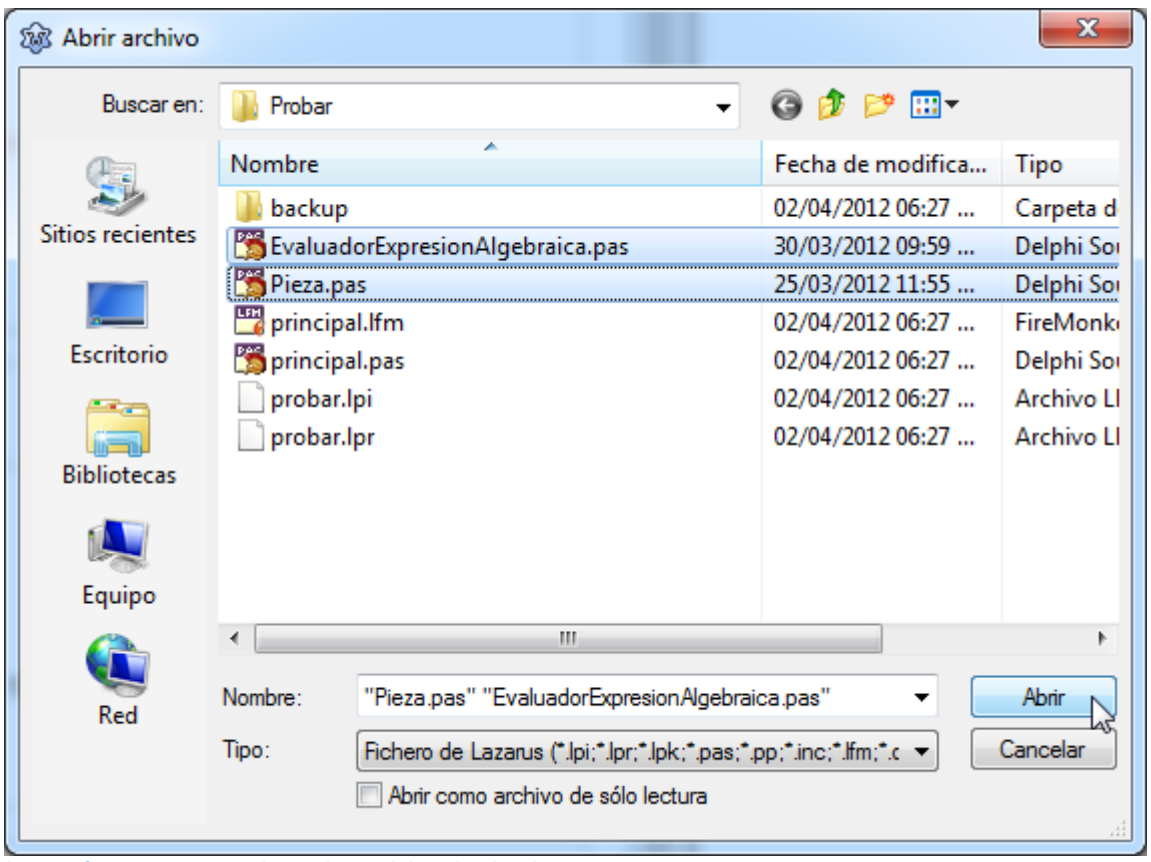


Ilustración 62: Se agregan los archivos del evaluador de expresiones

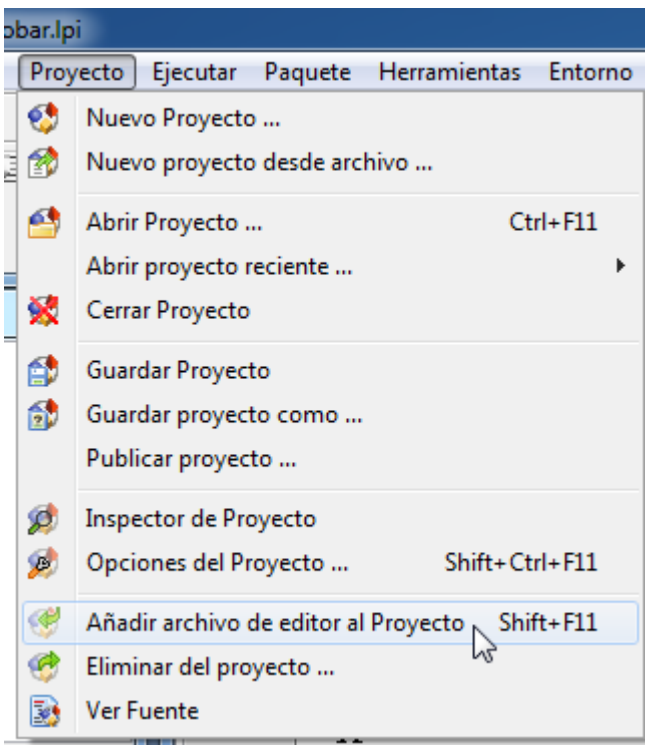


Ilustración 63: En Lazarus, una vez abierto el archivo se agregan al proyecto

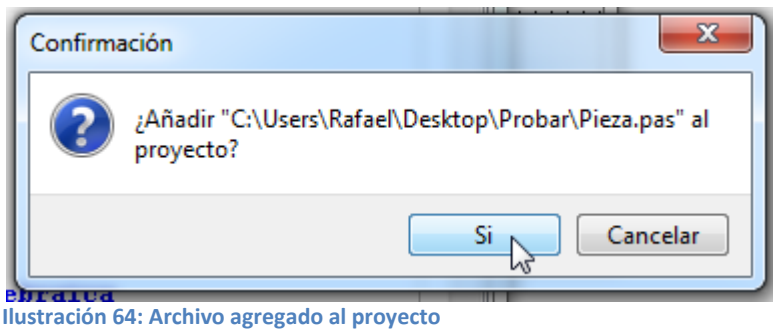


Ilustración 64: Archivo agregado al proyecto

Editor de código fuente

Pieza | EvaluadorExpresionAlgebraica | Principal

```
1  // Autor: Rafael Alberto Moreno Parra
.  // Correo: ramsoftware@gmail.com
.  // Página Web:  http://darwin.50webs.com
.  // Licencia: LGPL
5  // Fecha:  Marzo de 2012
.  unit EvaluadorExpresionAlgebraica;
7
.  {$MODE Delphi}
.  interface
10 uses
.    //Requerido para el TobjectList
.    Contnrs, Math, Pieza;
.
.  type
15   TEvaluadorExpresionAlgebraica = class
.    private
.        //Constantes para determinar que tipo es cada piez
```

Ilustración 65: Debe poner la directiva {$MODE Delphi} para que el código fuente compile en Lazarus

Editor de código fuente

Pieza | EvaluadorExpresionAlgebraica | Principal

```
1  //
.  // Autor: Rafael Alberto Moreno Parra
.  // Correo: ramsoftware@gmail.com
.  // Página Web: http://darwin.50webs.com
5  // Licencia: LGPL
.  // Fecha: Enero de 2012
.  //
.  unit Pieza;
.
10 {$MODE Delphi}
.  interface
.  type
.    TPieza = class
.    private
15     // Nivel en la expresión, a mayor valor, es mayor profun
.      nivel: integer;
.
.      // 1. Almacena un número, 2. Almacena un operador
.      tipo: integer;
20
.      // El número en la expresión algebraica */
.      numero: double;
```

Ilustración 66: Debe poner la directiva {$MODE Delphi} para que el código fuente compile en Lazarus

Editor de código fuente

Pieza | EvaluadorExpresionAlgebraica | Principal

```
.  { TfrmProbar }
.
.  procedure TfrmProbar.btnProbarClick(Sender: TObject);
.  var
35   evaluador: TEvaluadorExpresionAlgebraica;
.    resultado: double;
.  begin
.      evaluador :=  TEvaluadorExpresionAlgebraica.Create;
.      evaluador.Analizar('4.763*5.704+(x+sen(3.670/cos(sen(atn(sen(4.877*cos(7.330)))))))');
40     evaluador.DarValorVariable('x', 13.1729);
.      resultado := evaluador.Evaluar();
.      txtProbar.Text := FloattoStr(resultado);
.      evaluador.LiberaMemoria();
.  end;
45
.  end.
47
```

40: 49 | INS | C:\Users\Rafael\Desktop\Probar\principal.pas

Ilustración 67: Código para probar el evaluador de expresiones

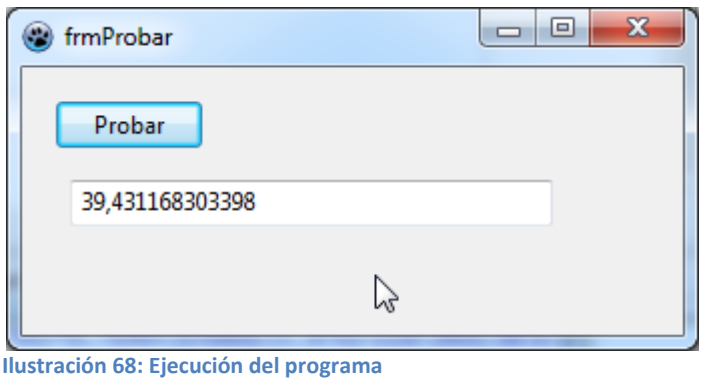


Ilustración 68: Ejecución del programa

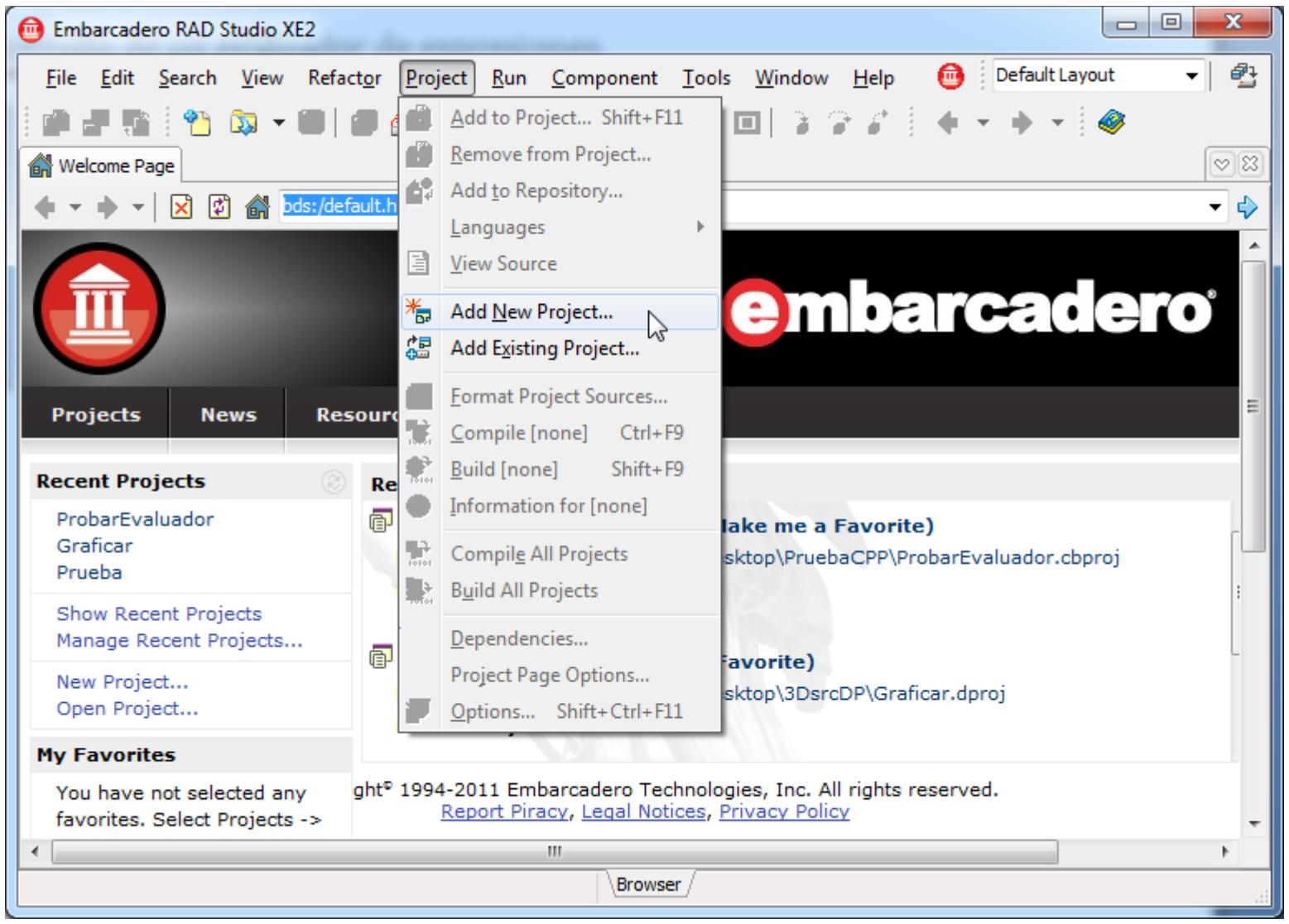


Ilustración 69: Se crea un nuevo proyecto en Embarcadero Delphi XE2

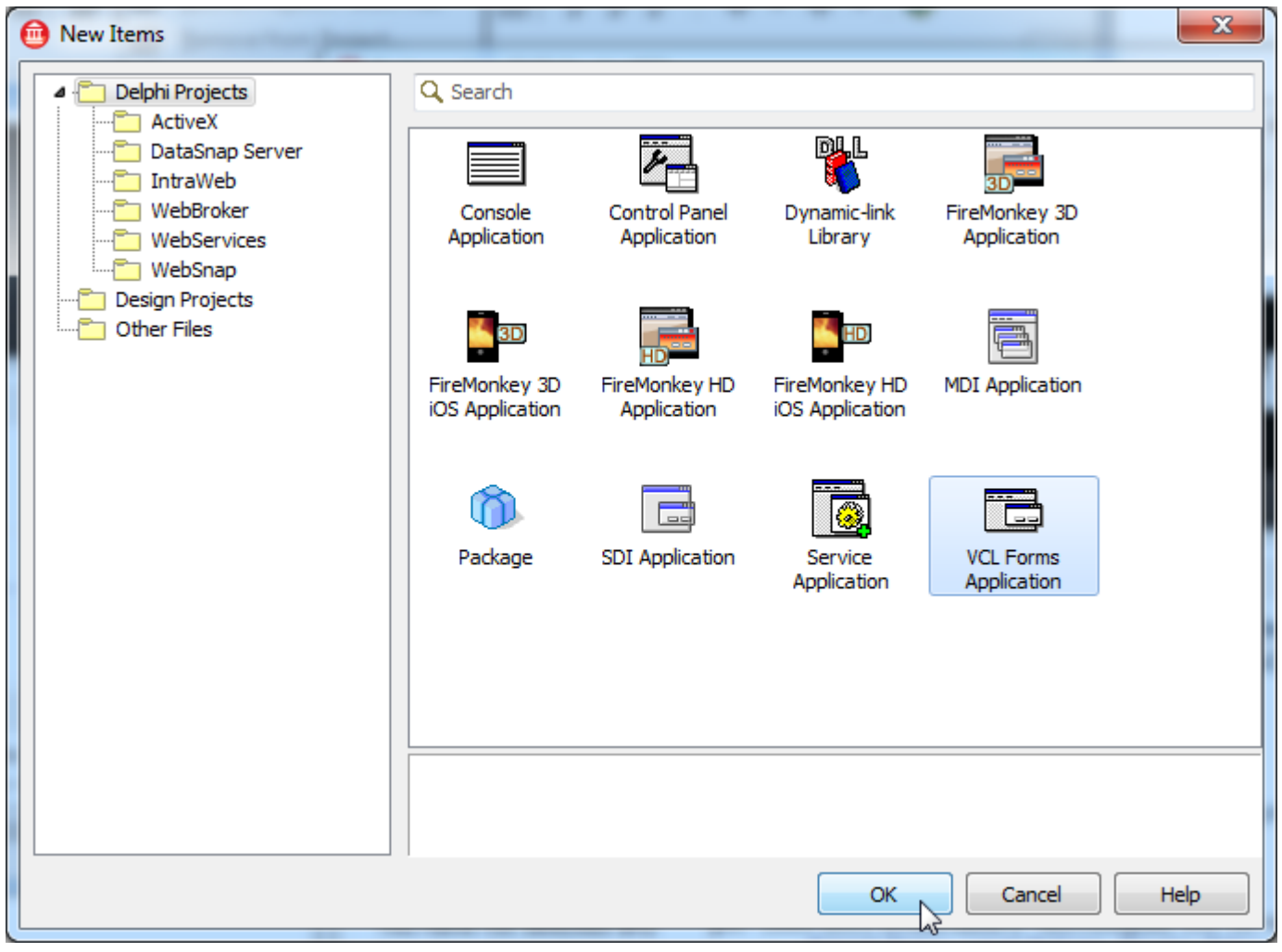


Ilustración 70: Una aplicación VCL

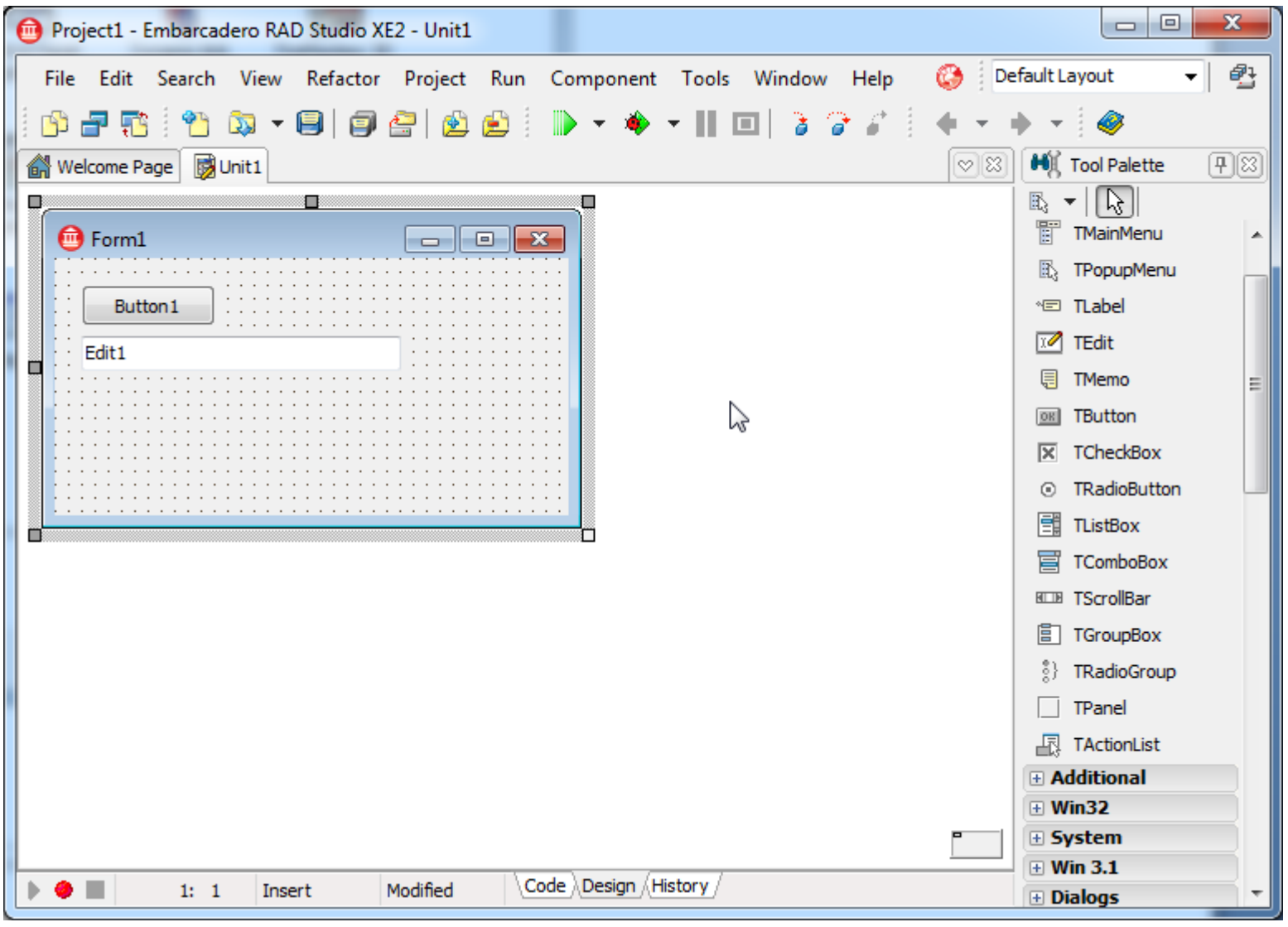


Ilustración 71: Un botón y una caja de texto

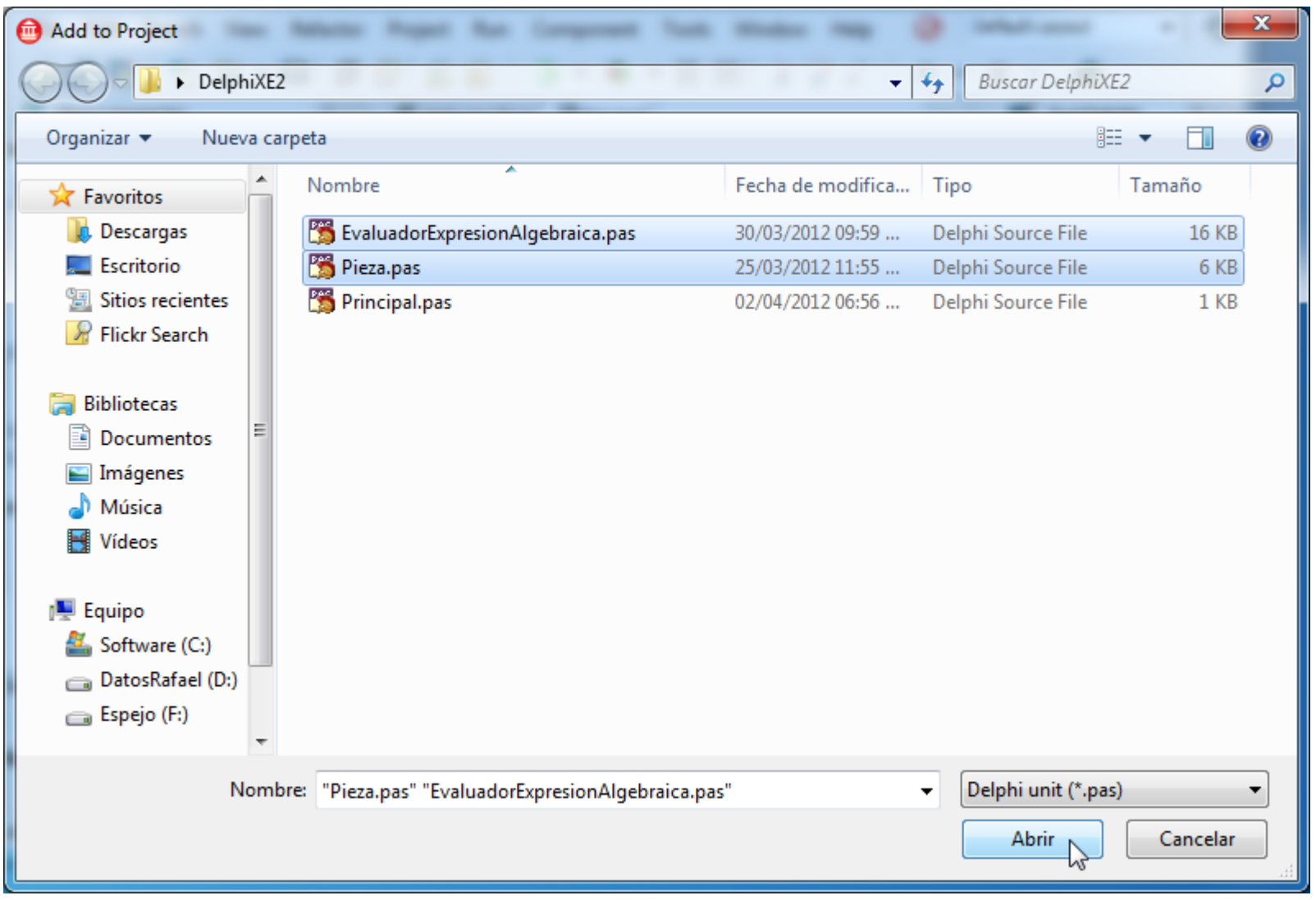


Ilustración 72: Se agregan los archivos del evaluador de expresiones

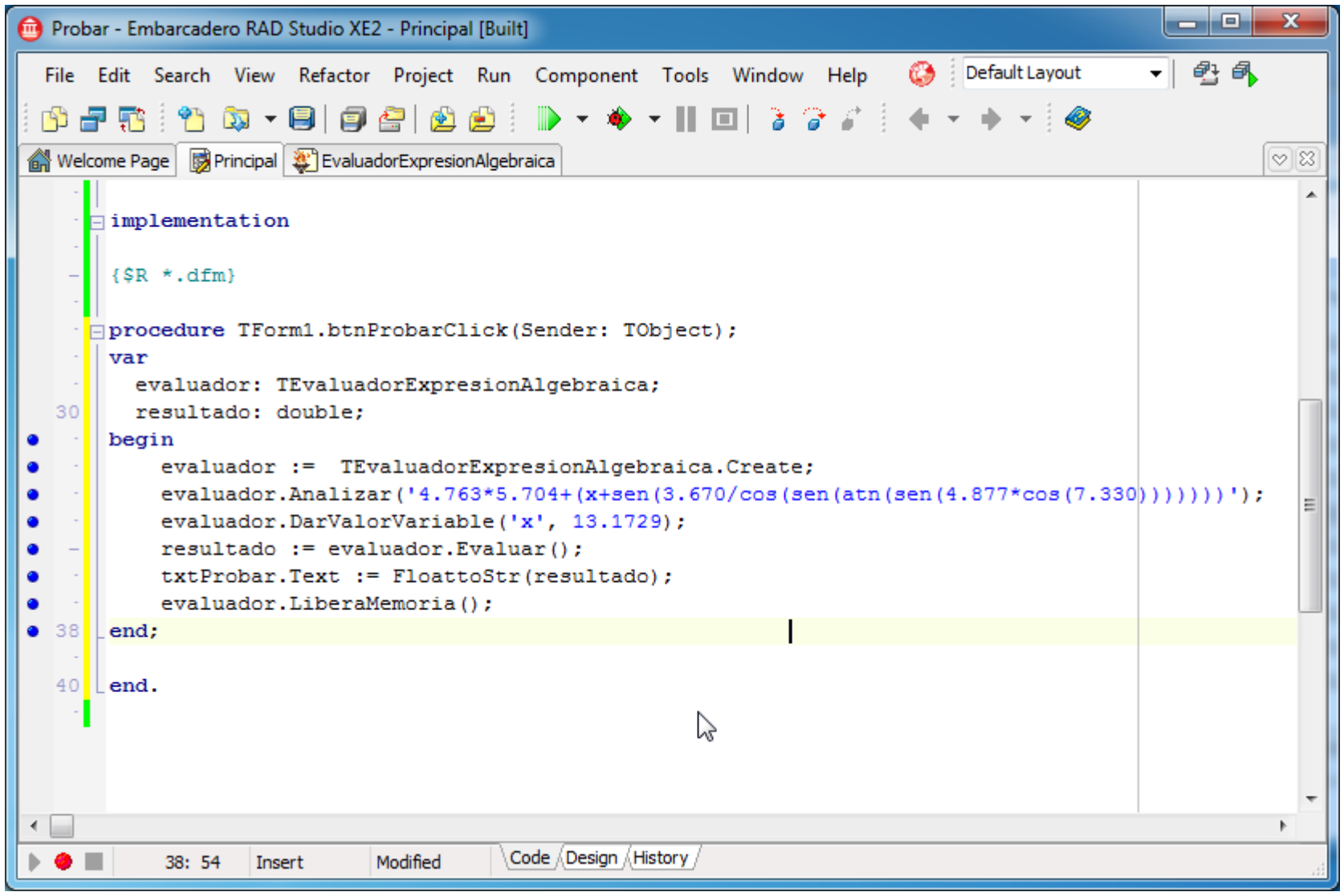


Ilustración 73: Código ejecutado por el botón para probar el evaluador de expresiones

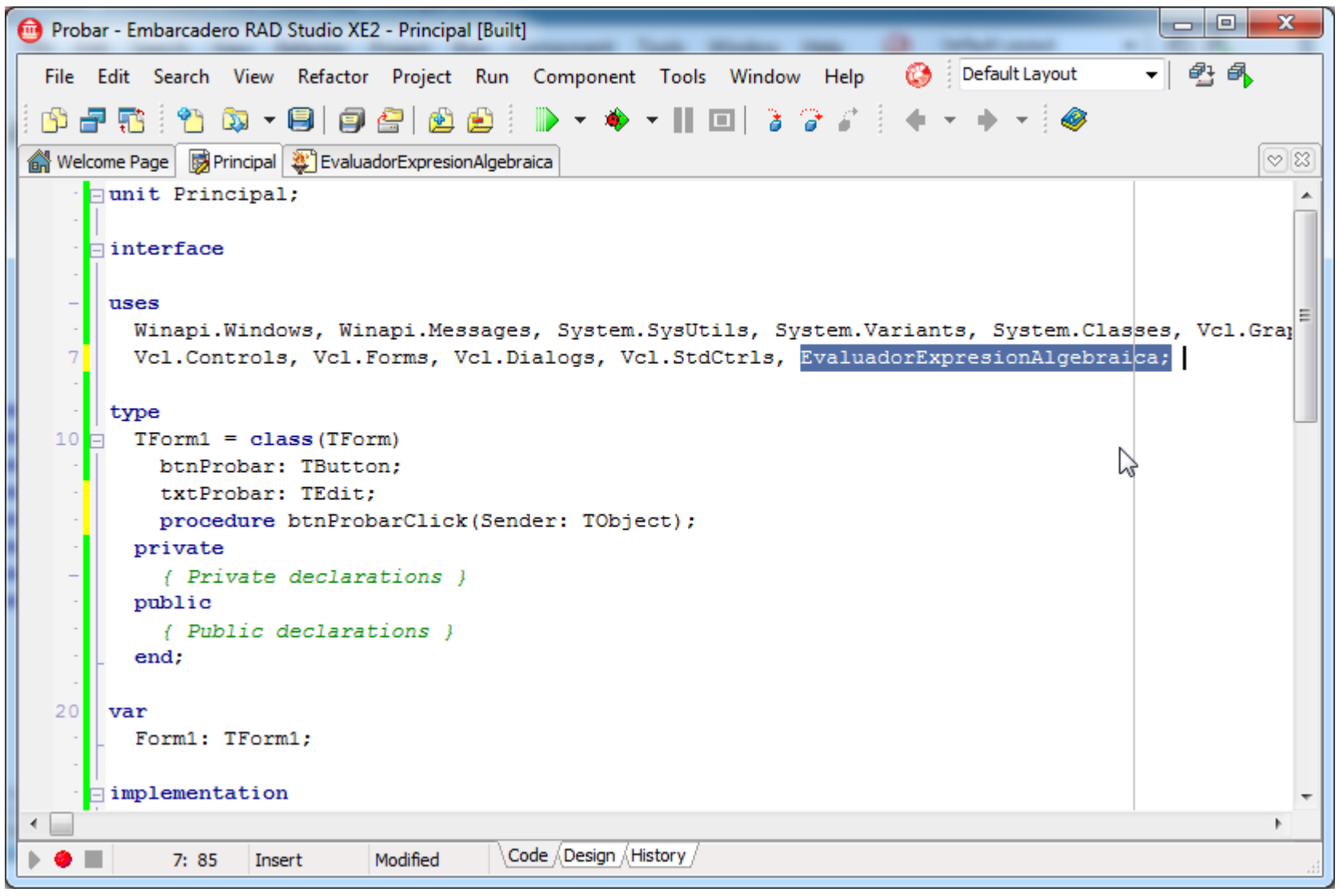


Ilustración 74: No olvidar hacer referencia de la clase del evaluador

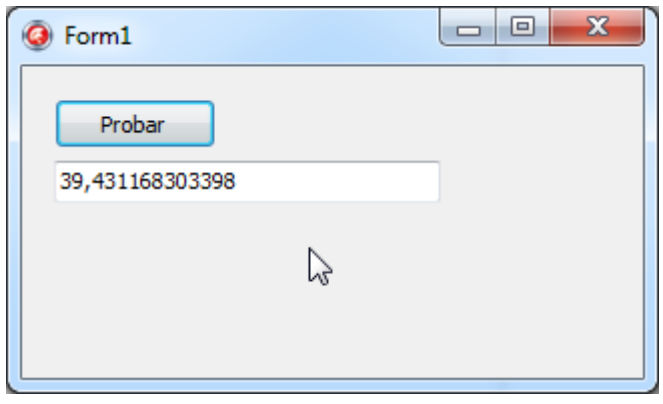


Ilustración 75: Ejecución del código

CONCLUSIONES

Un evaluador de expresiones le permite a uno conocer mucho las capacidades de un lenguaje de programación en cuanto al manejo de cadenas, estructura de datos, clases, objetos y sobre todo su desempeño. Los ejemplos aquí mostrados partieron de un mismo algoritmo.

Animo al lector a mejorar el evaluador de expresiones haciéndolo más rápido y con capacidad de manejar variables de varias letras. O hacerlo mas corto y menos complejo.

El evaluador mostrado aquí tiene dos partes: el analizador y el evaluador. El analizador convierte la expresión en una estructura que permite ser evaluada rápidamente. El evaluador recorre la estructura y evalúa la expresión con los nuevos valores de las variables algebraicas. Si desea hacer un software que dibuje en alto detalle la gráfica de una ecuación dada, entonces la parte del evaluador debe ser muy rápida porque sólo cambian los valores de las variables ya que la estructura de la ecuación permanece intacta. Pero si debe evaluar un gran número de ecuaciones y obtener un valor por cada una, entonces el analizador, es el que debe ser rápido.

El evaluador desarrollado en este libro, tiene balanceado ambos ítems en cuanto a velocidad.

El código fuente está listo para descarga aquí: http://darwin.50webs.com/Espanol/Evaluador.htm